ABng

Next Generation Active Badge System

J. Szymaszek, A. Uszok, K. Zieliński
e-mail:{jasz,uszok,kz}@ics.agh.edu.pl

Distributed Systems Research Group
Institute of Computer Science
University of Mining and Metallurgy in Cracow
Al. Mickiewicza 30, 30-059 Kraków, Poland

August 1997

1 Introduction

A new ABng–core internal model will be built taking into account experiences gathered during the existing core software implementation and will be based on following main assumptions:

1. Static components of the system that define its configuration, topology etc. will be separated from dymamic part that will be very light and event–processing oriented only,

2. The system will be structured according to the data flow paradigm.

3. The basic component of the system will be so called system view with uniform structure on dynamic level and semantics defined by static part.

4. The activity of the given view will be trigered by a sighting availability which will be passed rather by value than reference.

The structure of location system of new ABng–core system architecture is depicted in Fig. 1. The new ABng–core system architecture is based on the following terms:

- G_i – group of badge holders of view i

$$G_i \subseteq \{bh_1, bh_2, ...bh_n\} \tag{1}$$

  where bh_k- badge holder

- l_k – location k

$$l_k = \{s_1, s_2, ...s_m\} \tag{2}$$

  where s_l - sensor l

- L_i – set of locations of view i

$$L_i = \{l_1, l_2, ...l_r\} l_k \cap l_m = \emptyset \tag{3}$$

  for every k and m.

- V_i – view i defined as follows:

$$V_i : G_i \mapsto L_i \tag{4}$$

The view function defines a current state of the given view. It is modified by the sighting processing procedure. The sets and the function which form a View is maintained ed by the View Manager.

Two level sighting filtering procedure is employed in the core. First level is performed by **Main_Sighting_Processor**. The second level of sighting filtering is performed within view. Each view is built of the following uniforms elements:

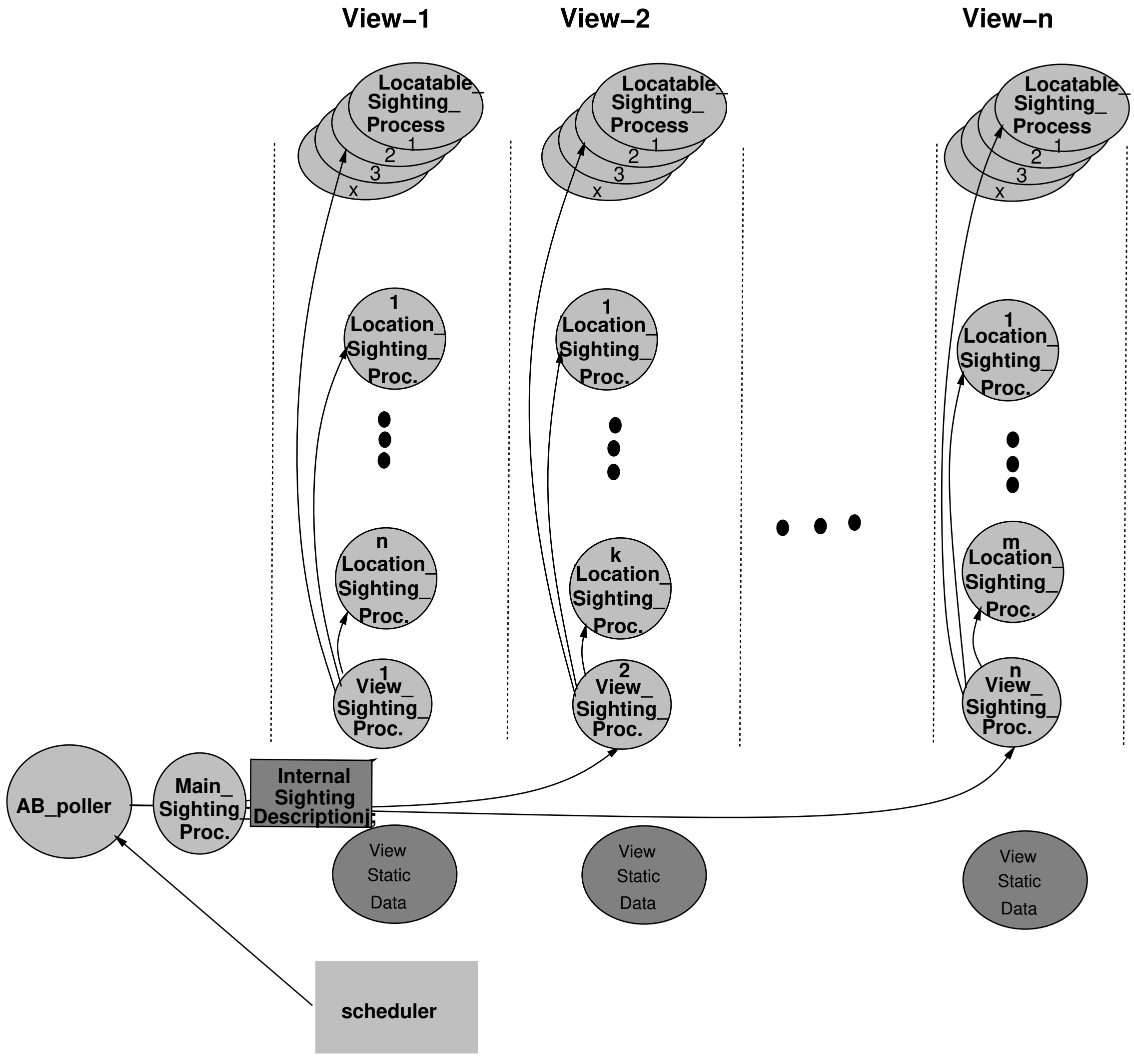

Figure 1: New ABng–core internal architecture

View_Sighting_Processor – its a routing element which routes sighting description to **Location_Sighting_Processors**. If the given sighting description contains sensor that are not present in the given view then the further processing is stopped.

Location_Sighting_Processor – modifies function & V_i and reports events to registered observers. Two **Location_Sighting_Processors** can be involved in that process – old and new one.

Badge_Sighting_Processor – holds the current location data of the given badge in the view. This processor is responsible for the relevant location data update.

2 Concept of the View

An office environment, in which the Active Badge System is installed and functioning, is seen by the ABng user through the concept of a **view**. It was introduced in order to structuralise and filter information collected by Active Badge System.

2.1 Construction of the view

A view contains of two sets of elements:

- **Location** – it is a part of a environment space obtained in a result of partition of the space performed according to an arbitrary user defined rule. This rule is called *Location_Type* and defines attributes associated with each of the location.
- **Locatable** – it is a object which is locatable within the space.

The given view could contain locations only of one *Location_Type*. A view possesses its unique, in the given domain, name. View, location and locatables concepts hide detail of Active Badge System configuration from client applications. They operate with abstract entities and do not deal with badges and sensors. The applications could even be not aware that the location information is gathered using active badges, it could easily be other localization system.

The precision of localization of locatable object within the space is equal to the size of locations of the view. A client can obtain information that either a given locatable from the view is within one of the location of this view or is outside all of them – *is outside this view*.

The same concerns invocations of commands by clients on locatables. A command can be send to a locatable only when it is inside one of locations of the view. A reply on this command will be delivered to the client only when it was received inside the view.

2.2 Security aspects

Every view existing in the system is associated with a list of users, which have access to this view. Thus only these users have access to localization information as well to static attributes describing locations and locatables from the view. It has been assumed that these users have access to this information as a whole – there are not additional assignment of permission concerning particular location or locatable.

The access to data is granted on the view level and it has to have effect on many other objects in other repositories associated with the given view. Thus the authorization server connected with the View Manager automatically grants a user access to all data about locations and locatables of the view. This authorization data is replicated in caches within repositories. Therefore when a user is denied access to the view or the contents of the view is changed (a location or locatable is removed from the view) the authorization server inform repositories caches about this changes.

The main concern was how to hidden the access control mechanism from the application code in order to make the system flexible. It was decided that the Principal mechanism will be employed in order to provide access control for ABng CORBA components without using the COSS Security Service, which is not transparent for applications. The Principal data contain authentication information and is automatically forwarded within all method calls. There are APIs on the ORB and on CORBA::Object which allows to manipulate the Principal. The schema of usage of the Principal is as follows:

1. A client application obtains the default Principal from the ORB.
2. The application ask the user to provide its identification information. Then it sets the Principal to this information. From that moment the Principal is included within every invocation.
3. On the server side there is an access control interceptor (Orbix filter) which intercepts an incoming call and extracts the Principal, the target object and the operation name. It now makes a call to authorization server and checks access rights.
4. Server proceed normally when access allowed.
5. Server raise standard *NO_PERMISSION* exception when access denied.

2.3 Proxy view

If ABng system is installed in the given domain then application from external domains could access information gathered by ABng, if they are permitted, using the standard view interface. However, very often it could be very inefficient. Thus the concept of proxy view was put forward. This component is a client of the view and uses the standard view interfaces. Internally, it creates proxies of view's locations and locatables as well as of all entities locations and locatables are related on and cache locally their states. Local client access proxy view by the usual view interfaces and get cached data. All object references returned by the view proxy point to real repositories objects proxies created inside the view proxy.

A local system administrator can configure a proxy view to provide information about only selected from the view sets locatables or location.

3 Uniform Model of Data Representation, Management and Searching

There is a lot of data, about different types of entities, which are required for ABng functionality as well as for applications built above it. These entities could be divided in two main categories:

- closely related to Active Badge System configuration,
- describing office environment in which Active Badge System was installed.

The entities are represented as a single out CORBA object, kept in the object repository adequate for them according to their type. The objects are made persisted using special Object Adapter.

An uniform template of all entity interfaces was defined:

```
#ifndef __ENTITY_IDL__
#define __ENTITY_IDL__ 1

interface Entity: Static_Data_Observed, Entity_Commander, ... {

    struct Description {
        Type1 attribute1;
        Type2 attribute2;
                .
                .
        TypeN attributeN;
    };

    typedef sequence<Description> Descriptions;

    struct Value_Description {
        Type1::Value_Description attribute1;
        Type2::Value_Description attribute2;
                .
                .
        TypeN::Value_Description attributeN;
    };

    typedef sequence<Value_Description> Value_Descriptions;


    struct Pattern {
        boolean is_any;
        Type1::Pattern attribute1_pattern;
        Type2::Pattern attribute2_pattern;
                .
                .
        TypeN::Pattern attributeN_pattern;
    };

    readonly attribute Repository_Item_Id item_id;
    readonly attribute Description descr;
    readonly attribute Type1 attribute1;
    readonly attribute Type2 attribute2;
                .
                .
    readonly attribute TypeN attributeN;
};

#endif                                              /* __ENTITY_IDL__ */
```

Each type of entity possesses a specific set of attributes: *attribute1*, ... ,*attributeN*. It also inherits from interface *Entity_Comander* which defines operation which could be invoked on it.

Each entity possesses a unique identifier *item_id* inside the repository.

Besides every entity inherits from the *Static_Data_Observed* interface (or in the case of entities related to Active Badge System configuration from *System_Static_Data_Observed*). This interface enables other objects to register in the entity its interest in changes of entity state. When the state is changed registered parties are informed.

Second uniform feature is a *descr* attribute of type *Description*, which is a structure possessing fields matching every attribute of the given entity. This structure is used to get the whole meaningful state of the entity in just one request. This functionality was designed in order to be used by the Smart Proxy mechanism, mainly. A smart proxy on the client side retrieves the whole state of the entity when the proxy is created and then serves local request using cached data. It also retrieves the whole state when cache is invalidated by the notification mechanism.

The next uniform element is a *Value_Description* structure. It is similar to the *Description* structure as it has field for every entity attribute but the type of these fields is either the type of the attribute, if it is not a reference, or the *Value_Description* structure from the entity referenced by this attribute. The purpose of this structure is to enable returning of the entity state as already collected all describing it data without any references to the outside objects.

Finally, there is a *Pattern* structure build, in a recursive way, of *Pattern* structure for simpler data types. Where this structure for the primitive type has the following form:

```
struct Pattern {
 boolean is_any;
 Primitive_Type primitive_type_pattern;
};
```

The *is_any* denote if the value of the *primitive_type_pattern* field is meaningful or not. The *Pattern* structure is used to specify searching criteria for the given type of entity.

Object build according to the template are stored in repositories which also posses interfaces build according to one pattern:

```
#ifndef __ENTITY_REP_IDL__
#define __ENTITY_REP_IDL__ 1

interface Entity_Rep : Static_Data_Observed {

    typedef sequence<Entity> Entitys;

    readonly attribute Entitys entity_list;

    Entity add(in Entity::Description data_record)
                        raises(Duplicate_Data_Record);
    void remove(in Entity object_to_remove)
                        raises(Unknown_Object_Ref);
    Entitys find(in Entity::Pattern data_record_pattern)
                        raises(Search_Not_Implemented);
    Entity::Value_Descriptions find_values(in Entity::Pattern data_record_pattern)
                        raises(Search_Not_Implemented);
    void update(in Entity object_to_update, in Entity::Description data_record_pattern)
                        raises(Unknown_Object_Ref, Duplicate_Data_Record);
};

#endif                                          /* __ENTITY_REP_IDL__ */
```

This interface enables:

- *Creation and addition* of the new entity to the repository with state described using *Description* structure. This method returns the object reference of the new entity. It also could raise the *Duplicate_Data_Record* when there already exists an entity with the same state.

- *Removing* of the existing entity by specifying its reference. It raises the *Unknown_Object_Ref* exception when such a entity does not exist.

- *Searching* of the entities matching specified criteria denoted by an adequate *Pattern* structure. The return value is either a sequence of object references (*find*) or sequence of *Value_Description* structures (*find_values*).

4 Database Layer

Data describing office environment considered as independent from the ABng CORE is stored in independent repositories and accessed via interfaces from separate modules:

- *Domain_Repository* – interfaces from this module enable only reading of data,

```
#include <orb.idl>

module Domain_Repository {

typedef sequence < short > Shorts ;
typedef sequence < long > Longs ;
typedef sequence < unsigned short > Ushorts ;
typedef sequence < unsigned long > Ulongs ;
typedef sequence < float > Floats ;
typedef sequence < double > Doubles ;
typedef sequence < char > Chars ;
typedef sequence < boolean > Booleans ;

interface Static_Data_Observer ;
interface Static_Data_Observed_E ;
interface Static_Data_Observed_R ;
interface Location_Type ;
interface Location_Type_Rep ;
interface Location_Description ;
interface Location_Description_Rep ;
interface User ;
interface User_Rep ;
interface Meta_Equipment ;
interface Meta_Equipment_Rep ;
interface Equipment ;
interface Equipment_Rep ;



struct Property_Description {
string name ;
```

```
TypeCode type ;
} ;

typedef sequence < Property_Description > Property_Descriptions ;

struct Property {
string name ;
any value ;
} ;

typedef sequence < Property > Properties ;

typedef sequence < string > Strings ;

struct Time {
unsigned long seconds ;
unsigned long microseconds ;
} ;

typedef sequence < octet > Octets ;

typedef unsigned long Repository_Item_Id ;
typedef sequence < Repository_Item_Id > Repository_Item_Ids ;

struct String_Pattern {
boolean is_any ;
string value ;
} ;

struct Octets_Pattern {
boolean is_any ;
Octets value ;
} ;

typedef sequence < String_Pattern > String_Patterns ;

enum Sequence_Pattern_Constraints {
UNRESTRICTED ,
RESTRICTED_ORDERING ,
RESTRICTED_NUMBER ,
RESTRICTED_ORDERING_AND_NUMBER
} ;

struct Strings_Pattern {
boolean is_any ;
Sequence_Pattern_Constraints constraints ;
String_Patterns value ;
} ;

struct TypeCode_Pattern {
boolean is_any ;
TypeCode value ;
```

```
} ;

struct Property_Description_Pattern {
boolean is_any ;
String_Pattern name_pattern ;
TypeCode_Pattern type_pattern ;
} ;

typedef sequence < Property_Description_Pattern > Property_Description_Patterns ;

struct Property_Descriptions_Pattern {
boolean is_any ;
Sequence_Pattern_Constraints constraints ;
Property_Description_Patterns value ;
} ;

struct Any_Pattern {
boolean is_any ;
any value ;
} ;

struct Property_Pattern {
boolean is_any ;
String_Pattern name_pattern ;
Any_Pattern value_pattern ;
} ;

typedef sequence < Property_Pattern > Property_Patterns ;

struct Properties_Pattern {
boolean is_any ;
Sequence_Pattern_Constraints constraints ;
Property_Patterns value ;
} ;

struct Unsigned_Long_Pattern {
boolean is_any ;
unsigned long value ;
} ;

typedef Unsigned_Long_Pattern Repository_Item_Id_Pattern ;

struct Unsigned_Short_Pattern {
boolean is_any ;
unsigned short value ;
} ;

struct Boolean_Pattern {
boolean is_any ;
boolean value ;
} ;
```

```
enum Change_Type {
OBJECT_ADDED ,
OBJECT_REMOVED ,
OBJECT_UPDATED
} ;

exception Duplicate_Data_Record { } ;
exception Unknown_Object_Ref { } ;
exception Search_Not_Implemented { } ;
exception Unknown_Observer { } ;
exception Invalid_Badge_Holder { } ;
exception Unknown_Repository_Item_Ids {
Repository_Item_Ids unknown_items ;
} ;
exception Unmatched_Observer_Item_Ids {
Repository_Item_Ids unknown_items ;
} ;
exception Destination_Unreachable {
string server_name ;
} ;
exception Operation_Not_Allowed { } ;
exception Invalid_Description { } ;

interface Static_Data_Observed_E {

typedef sequence < Static_Data_Observer > Static_Data_Observers ;
readonly attribute Static_Data_Observers static_data_observer_list ;

void attach_static_data_observer (
in Static_Data_Observer obs ) ;

void detach_static_data_observer (
in Static_Data_Observer obs )
raises ( Unknown_Observer ) ;
} ;

interface Static_Data_Observed_R {

struct Observer_Entry {
Static_Data_Observer obs ;
Repository_Item_Ids interest ;
} ;

typedef sequence < Observer_Entry > Observer_Entries ;
readonly attribute Observer_Entries static_data_observer_list ;

readonly attribute Static_Data_Observed_E :: Static_Data_Observers
add_static_data_observer_list ;

void attach_static_data_observer (
in Static_Data_Observer obs ,
in Repository_Item_Ids observed_id_list )
```

```
raises ( Unknown_Repository_Item_Ids ) ;

void detach_static_data_observer (
in Static_Data_Observer obs ,
in Repository_Item_Ids observed_id_list )
raises ( Unknown_Repository_Item_Ids , Unknown_Observer ,
Unmatched_Observer_Item_Ids ) ;

void attach_add_static_data_observer (
in Static_Data_Observer obs ) ;

void detach_add_static_data_observer (
in Static_Data_Observer obs )
raises ( Domain_Repository :: Unknown_Observer ) ;
} ;

interface Static_Data_Observer {

enum Static_Data_Type {
USER_SD ,
META_EQUIPMENT_SD ,
EQUIPMENT_SD ,
LOCATION_TYPE_SD ,
LOCATION_DESCRIPTION_SD
} ;

union Static_Data_Object switch ( Static_Data_Type ) {
case USER_SD : User user_obj ;
case META_EQUIPMENT_SD : Meta_Equipment meta_equipment_obj ;
case EQUIPMENT_SD : Equipment equipment_obj ;
case LOCATION_TYPE_SD : Location_Type location_type_obj ;
case LOCATION_DESCRIPTION_SD : Location_Description location_descr_obj ;
} ;

oneway void notify (
in Domain_Repository :: Repository_Item_Id id ,
in Static_Data_Object sd ,
in Domain_Repository :: Change_Type change ) ;
oneway void abort ( ) ;
} ;

interface Local_Dependence {

struct Telephone_Dependence {
string international_access_code ;
string country_access_code ;
string intercity_access_code ;
string city_access_code ;
string local_extension ;
} ;

typedef sequence < string > Email_Dependence ;
```

```
readonly attribute Telephone_Dependence my_telephone_dependence ;
readonly attribute Email_Dependence my_email_dependence ;
readonly attribute Time GMT_interval ;
} ;

interface Location_Type : Static_Data_Observed_E {

typedef string Location_Type_Name ;
typedef String_Pattern Location_Type_Name_Pattern ;

struct Description {
Repository_Item_Id item_id ;
Location_Type_Name name ;
Property_Descriptions property_description_list ;
} ;

typedef sequence < Description > Descriptions ;

typedef Description Value_Description ;
typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
Repository_Item_Id_Pattern item_id_pattern ;
Location_Type_Name_Pattern name_pattern ;
Property_Descriptions_Pattern
property_description_list_pattern ;
} ;

readonly attribute Description descr ;
readonly attribute Value_Description value_descr ;
readonly attribute Repository_Item_Id item_id ;
readonly attribute Location_Type_Name name ;
readonly attribute Property_Descriptions property_description_list ;
} ;

interface Location_Type_Rep : Static_Data_Observed_R {

typedef sequence < Location_Type > Location_Types ;

Location_Types find (
in Location_Type :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ) ;
Location_Type :: Value_Descriptions find_values (
in Location_Type :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ) ;
} ;

interface Location_Description : Static_Data_Observed_E {

struct Description {
```

```
Repository_Item_Id item_id ;
Location_Type :: Location_Type_Name type_name ;
Properties property_list ;
} ;

typedef sequence < Description > Descriptions ;

typedef Description Value_Description ;
typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
Repository_Item_Id_Pattern item_id_pattern ;
Location_Type :: Location_Type_Name_Pattern type_name_pattern ;
Properties_Pattern property_list_pattern ;
} ;

readonly attribute Description descr ;
readonly attribute Value_Description value_descr ;
readonly attribute Repository_Item_Id item_id ;



readonly attribute string type_name ;

readonly attribute Properties property_list ;
} ;

interface Location_Description_Rep : Static_Data_Observed_R {

typedef sequence < Location_Description > Location_Descriptions ;

Location_Descriptions find (
in Location_Description :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ) ;
Location_Description :: Value_Descriptions find_values (
in Location_Description :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ) ;
} ;

interface User : Static_Data_Observed_E {

struct Description {
Repository_Item_Id item_id ;
string uid ;
string forename ;
string surname ;
string position ;
string email ;
string phone ;
} ;
```

```
typedef sequence < Description > Descriptions ;

typedef Description Value_Description ;
typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
Repository_Item_Id_Pattern item_id_pattern ;
String_Pattern uid_pattern ;
String_Pattern forename_pattern ;
String_Pattern surname_pattern ;
String_Pattern position_pattern ;
String_Pattern email_pattern ;
String_Pattern phone_pattern ;
} ;

readonly attribute Description descr ;
readonly attribute Value_Description value_descr ;
readonly attribute Repository_Item_Id item_id ;
readonly attribute string uid ;
readonly attribute string forename ;
readonly attribute string surname ;
readonly attribute string position ;
readonly attribute string email ;
readonly attribute string phone ;
} ;

interface User_Rep : Static_Data_Observed_R {

typedef sequence < User > Users ;

Users find ( in User :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ) ;
User :: Value_Descriptions find_values (
in User :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ) ;
} ;

typedef Strings Command_List ;
typedef Strings_Pattern Command_List_Pattern ;
typedef Octets IR_Data ;
typedef Octets_Pattern IR_Data_Pattern ;

struct IR_Control_Data {
Command_List commands ;
IR_Data my_IR_data ;
} ;

struct IR_Control_Data_Pattern {
boolean is_any ;
Command_List_Pattern commands_pattern ;
IR_Data_Pattern my_IR_data_pattern ;
```

```
} ;

interface Meta_Equipment : Static_Data_Observed_E {
typedef Octets Image ;

typedef Octets_Pattern Image_Pattern ;

struct Dimensions {
unsigned long height ;
unsigned long width ;
unsigned long depth ;
} ;

struct Dimensions_Pattern {
boolean is_any ;
Unsigned_Long_Pattern height_pattern ;
Unsigned_Long_Pattern width_pattern ;
Unsigned_Long_Pattern depth_pattern ;
} ;

enum Control_Kind { INFRA_RED , ANYTHING_ELSE } ;

union Control_Info switch ( Control_Kind ) {
case INFRA_RED : IR_Control_Data my_IR_control_data ;
default : unsigned long default_field ;
} ;

struct Control_Info_Pattern {
boolean is_any ;
union IRCI_Pattern switch ( Control_Kind ) {
case INFRA_RED : IR_Control_Data_Pattern my_IRCD_pattern ;
default : Unsigned_Long_Pattern default_field_pattern ;
} my_control_data_pattern ;
} ;

struct Description {
Repository_Item_Id item_id ;

string type ;

string vendor ;
string model ;

Dimensions my_dimensions ;

Control_Info my_control_info ;

Property_Descriptions
specific_property_description_list ;

Image my_icon ;
} ;
```

```
typedef sequence < Description > Descriptions ;

typedef Description Value_Description ;
typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
Repository_Item_Id_Pattern item_id_pattern ;

boolean is_any ;

String_Pattern type_pattern ;
String_Pattern vendor_pattern ;
String_Pattern model_pattern ;

Dimensions_Pattern my_dimensions_pattern ;

Control_Info_Pattern my_control_info_pattern ;

Property_Descriptions_Pattern
specific_property_description_list_pattern ;

Image_Pattern my_icon_pattern ;
} ;

readonly attribute Description descr ;
readonly attribute Value_Description value_descr ;
readonly attribute Repository_Item_Id item_id ;
readonly attribute string type ;
readonly attribute string vendor ;
readonly attribute string model ;
readonly attribute Dimensions my_dimensions ;
readonly attribute Control_Info my_control_info ;
readonly attribute Property_Descriptions
specific_property_description_list ;
readonly attribute Image my_icon ;
} ;

interface Meta_Equipment_Rep : Static_Data_Observed_R {

typedef sequence < Meta_Equipment > Meta_Equipments ;

Meta_Equipments find (
in Meta_Equipment :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ) ;
Meta_Equipment :: Value_Descriptions find_values (
in Meta_Equipment :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ) ;
} ;

interface Equipment : Static_Data_Observed_E {
```

```
struct Description {
Repository_Item_Id item_id ;
Meta_Equipment meta_data ;
string name ;
string last_command ;
Properties property_list ;
} ;

typedef sequence < Description > Descriptions ;

struct Value_Description {
Repository_Item_Id item_id ;
Meta_Equipment :: Description meta_data ;
string name ;
string last_command ;
Properties property_list ;
} ;
typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
Repository_Item_Id_Pattern item_id_pattern ;
Meta_Equipment :: Pattern meta_data_pattern ;
String_Pattern name_pattern ;
String_Pattern last_command_pattern ;
Properties_Pattern property_list_pattern ;
} ;

Description descr ( )
raises ( Destination_Unreachable ) ;

Value_Description value_descr ( )
raises ( Destination_Unreachable ) ;

readonly attribute Repository_Item_Id item_id ;

Meta_Equipment meta_data ( )
raises ( Destination_Unreachable ) ;

readonly attribute string name ;
readonly attribute string last_command ;
readonly attribute Properties property_list ;

} ;

interface Equipment_Rep : Static_Data_Observed_R {

typedef sequence < Equipment > Equipments ;

Equipments find ( in Equipment :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ,
Destination_Unreachable ) ;
```

```
Equipment :: Value_Descriptions find_values (
in Equipment :: Pattern data_record_pattern )
raises ( Search_Not_Implemented ,
Destination_Unreachable ) ;
} ;

interface Domain_DB {

readonly attribute Location_Type_Rep my_location_type_rep ;
readonly attribute Location_Description_Rep my_location_description_rep ;
readonly attribute User_Rep my_user_rep ;
readonly attribute Equipment_Rep my_equipment_rep ;
} ;

interface Domain {

readonly attribute Location_Description my_location_description ;
readonly attribute Local_Dependence my_local_dependence ;
readonly attribute Domain_DB my_domain_db ;
} ;

} ;
```

- *Domain_Repository_Management* – interfaces from this module are intended to be used by a system administrator thus they enable not only reading but writing too,

```
#include <orb.idl>

#include "domain_rep.idl"

module Domain_Repository_Management {

interface Location_Type_Rep_Manager {

readonly attribute Domain_Repository :: Location_Type_Rep rep ;
readonly attribute Domain_Repository :: Location_Type_Rep :: Location_Types
location_type_list ;

Domain_Repository :: Location_Type add (
in Domain_Repository :: Location_Type :: Description data_record )
raises ( Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Invalid_Description ) ;
void remove ( in Domain_Repository :: Location_Type object_to_remove )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Domain_Repository :: Operation_Not_Allowed ) ;
void update ( in Domain_Repository :: Location_Type object_to_update ,
in Domain_Repository :: Location_Type :: Description data_record_pattern )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Invalid_Description ) ;
} ;

interface Location_Description_Rep_Manager {
```

```
readonly attribute Domain_Repository :: Location_Description_Rep rep ;
readonly attribute
Domain_Repository :: Location_Description_Rep :: Location_Descriptions
location_description_list ;

Domain_Repository :: Location_Description add (
in Domain_Repository :: Location_Description :: Description data_record )
raises ( Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Invalid_Description ) ;
void remove ( in Domain_Repository :: Location_Description object_to_remove )
raises ( Domain_Repository :: Unknown_Object_Ref ) ;
void update ( in Domain_Repository :: Location_Description object_to_update ,
in Domain_Repository :: Location_Description :: Description
data_record_pattern )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Invalid_Description ) ;
} ;

interface User_Rep_Manager {

readonly attribute Domain_Repository :: User_Rep rep ;
readonly attribute Domain_Repository :: User_Rep :: Users user_list ;

Domain_Repository :: User add (
in Domain_Repository :: User :: Description data_record )
raises ( Domain_Repository :: Duplicate_Data_Record ) ;

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

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

interface Meta_Equipment_Rep_Manager {

readonly attribute Domain_Repository :: Meta_Equipment_Rep rep ;
readonly attribute Domain_Repository :: Meta_Equipment_Rep :: Meta_Equipments
meta_equipment_list ;

Domain_Repository :: Meta_Equipment add (
in Domain_Repository :: Meta_Equipment :: Description data_record )
raises ( Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Invalid_Description ) ;

void remove ( in Domain_Repository :: Meta_Equipment object_to_remove )
```

```
raises ( Domain_Repository :: Unknown_Object_Ref ,
Domain_Repository :: Operation_Not_Allowed ) ;

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

interface Equipment_Rep_Manager {

readonly attribute Domain_Repository :: Equipment_Rep rep ;
readonly attribute Domain_Repository :: Equipment_Rep :: Equipments
equipment_list ;

Domain_Repository :: Equipment add ( in
Domain_Repository :: Equipment :: Description data_record )
raises ( Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Invalid_Description ,
Domain_Repository :: Destination_Unreachable ) ;
void remove ( in Domain_Repository :: Equipment object_to_remove )
raises ( Domain_Repository :: Unknown_Object_Ref ) ;
void update ( in Domain_Repository :: Equipment object_to_update ,
in Domain_Repository :: Equipment :: Description data_record_pattern )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Invalid_Description ,
Domain_Repository :: Destination_Unreachable ) ;
} ;

interface Domain_DB_Manager {

readonly attribute Location_Type_Rep_Manager
my_location_type_rep_manager ;
readonly attribute Location_Description_Rep_Manager
my_location_description_rep_manager ;
readonly attribute User_Rep_Manager
my_user_rep_manager ;
readonly attribute Equipment_Rep_Manager
my_equipment_rep_manager ;
} ;

interface Domain_Manager {

readonly attribute Domain_Repository :: Domain my_domain ;
attribute Domain_Repository :: Location_Description my_location_description ;
attribute Domain_Repository :: Local_Dependence my_local_dependence ;
attribute Domain_DB_Manager my_domain_db_manager ;
} ;
```

```
} ;
```

Common data types for all interfaces from these module are:

```
#ifndef __DOMAIN_TYPES_IDL__
#define __DOMAIN_TYPES_IDL__ 1


//interface TypeCode {};

typedef sequence<short> Shorts;
typedef sequence<long> Longs;
typedef sequence<unsigned short> Ushorts;
typedef sequence<unsigned long> Ulongs;
typedef sequence<float> Floats;
typedef sequence<double> Doubles;
typedef sequence<char> Chars;
typedef sequence<boolean> Booleans;

interface Static_Data_Observer;
interface Static_Data_Observed_E;
interface Static_Data_Observed_R;
interface Location_Type;
interface Location_Type_Rep;
interface Location_Description;
interface Location_Description_Rep;
interface User;
interface User_Rep;
interface Meta_Equipment;
interface Meta_Equipment_Rep;
interface Equipment;
interface Equipment_Rep;

// Common type definitions:

struct Property_Description {
    string name;
    TypeCode type;
};

typedef sequence<Property_Description> Property_Descriptions;

struct Property {
    string name;
    any value;
};

typedef sequence<Property> Properties;

typedef sequence<string> Strings;

struct Time {
    unsigned long seconds;
```

```
    unsigned long microseconds;
};

typedef sequence<octet> Octets;

typedef unsigned long Repository_Item_Id;
typedef sequence<Repository_Item_Id> Repository_Item_Ids;

struct String_Pattern {
    boolean is_any;
    string value;
};

struct Octets_Pattern {
    boolean is_any;
    Octets value;
};

typedef sequence<String_Pattern> String_Patterns;

enum Sequence_Pattern_Constraints {
    UNRESTRICTED,
    RESTRICTED_ORDERING,
    RESTRICTED_NUMBER,
    RESTRICTED_ORDERING_AND_NUMBER
};

struct Strings_Pattern {
    boolean is_any;
    Sequence_Pattern_Constraints constraints;
    String_Patterns value;
};

struct TypeCode_Pattern {
    boolean is_any;
    TypeCode value;
};

struct Property_Description_Pattern {
    boolean is_any;
    String_Pattern name_pattern;
    TypeCode_Pattern type_pattern;
};

typedef sequence<Property_Description_Pattern> Property_Description_Patterns;

struct Property_Descriptions_Pattern {
    boolean is_any;
    Sequence_Pattern_Constraints constraints;
    Property_Description_Patterns value;
};
```

```
struct Any_Pattern {
    boolean is_any;
    any value;
};

struct Property_Pattern {
    boolean is_any;
    String_Pattern name_pattern;
    Any_Pattern value_pattern;
};

typedef sequence<Property_Pattern> Property_Patterns;

struct Properties_Pattern {
    boolean is_any;
    Sequence_Pattern_Constraints constraints;
    Property_Patterns value;
};

struct Unsigned_Long_Pattern {
    boolean is_any;
    unsigned long value;
};

typedef Unsigned_Long_Pattern Repository_Item_Id_Pattern;

struct Unsigned_Short_Pattern {
    boolean is_any;
    unsigned short value;
};

struct Boolean_Pattern {
    boolean is_any;
    boolean value;
};

enum Change_Type {
    OBJECT_ADDED,
    OBJECT_REMOVED,
    OBJECT_UPDATED
};

exception Duplicate_Data_Record {};
exception Unknown_Object_Ref {};
exception Search_Not_Implemented {};
exception Unknown_Observer {};
exception Invalid_Badge_Holder {};
exception Unknown_Repository_Item_Ids {
    Repository_Item_Ids unknown_items;
};
exception Unmatched_Observer_Item_Ids {
    Repository_Item_Ids unknown_items;
```

```
};
exception Destination_Unreachable {
    string server_name;
};
exception Operation_Not_Allowed {};
exception Invalid_Description {};


#endif                                                  /* __DOMAIN_TYPES_IDL__ */
```

4.1 Domain

The root for all repositories containing information about the given domain, not specific for Abng, was called *Domain*. There are two interface to its state:

- enabling only reading of attributes:

```
#ifndef __DOMAIN_IDL__
#define __DOMAIN_IDL__ 1

interface Domain {

    readonly attribute Location_Description my_location_description;
    readonly attribute Local_Dependence my_local_dependence;
    readonly attribute Domain_DB my_domain_db;
};

#endif                                                  /* __DOMAIN_IDL__ */
```

- enabling reading and making changes to attributes:

```
#ifndef __DOMAIN_MANAGER_IDL__
#define __DOMAIN_MANAGER_IDL__ 1

interface Domain_Manager {

    readonly attribute Domain_Repository::Domain my_domain;
    attribute Domain_Repository::Location_Description my_location_description;
    attribute Domain_Repository::Local_Dependence my_local_dependence;
    attribute Domain_DB_Manager my_domain_db_manager;
};

#endif                                                  /* __DOMAIN_MANAGER_IDL__ */
```

Its states was divided into three sections:

- list of references to repositories described in this chapter (*my_domain_db*):

```
#ifndef __DOMAIN_DB_IDL__
#define __DOMAIN_DB_IDL__ 1

interface Domain_DB {
```

```
    readonly attribute Location_Type_Rep my_location_type_rep;
    readonly attribute Location_Description_Rep my_location_description_rep;
    readonly attribute User_Rep my_user_rep;
    readonly attribute Equipment_Rep my_equipment_rep;
};

#endif                                          /* __DOMAIN_DB_IDL__ */
```

or in *Domain_Manager* list of references to repositories managers (*my_domain_db_manager*):

```
#ifndef __DOMAIN_DB_MANAGER_IDL__
#define __DOMAIN_DB_MANAGER_IDL__ 1

interface Domain_DB_Manager {

    readonly attribute Location_Type_Rep_Manager
        my_location_type_rep_manager;
    readonly attribute Location_Description_Rep_Manager
        my_location_description_rep_manager;
    readonly attribute User_Rep_Manager
        my_user_rep_manager;
    readonly attribute Equipment_Rep_Manager
        my_equipment_rep_manager;
};

#endif                                  /* __DOMAIN_DB_MANAGER_IDL__ */
```

- list of properties describing domain according to the *Location Types Repository*,
- information about local dependences, which are stored in the following object:

```
#ifndef __LOCAL_DEPENDENCE_IDL__
#define __LOCAL_DEPENDENCE_IDL__ 1

interface Local_Dependence {

    struct Telephone_Dependence {
        string international_access_code;
        string country_access_code;
        string intercity_access_code;
        string city_access_code;
        string local_extension;
    };

    typedef sequence<string> Email_Dependence;

    readonly attribute Telephone_Dependence my_telephone_dependence;
    readonly attribute Email_Dependence my_email_dependence;
    readonly attribute Time GMT_interval;
};

#endif                                  /* __LOCAL_DEPENDENCE_IDL__ */
```

4.2 Interfaces of notification mechanism

All repositories and entities stored in them provide interface enabling clients to register their interest in changes of repositories and entities states:

```
#ifndef __STATIC_DATA_OBSERVER_IDL__
#define __STATIC_DATA_OBSERVER_IDL__ 1

interface Static_Data_Observer {

    enum Static_Data_Type {
        USER_SD,
        META_EQUIPMENT_SD,
        EQUIPMENT_SD,
        LOCATION_TYPE_SD,
        LOCATION_DESCRIPTION_SD
    };

    union Static_Data_Object switch (Static_Data_Type) {
        case USER_SD : User user_obj;
        case META_EQUIPMENT_SD : Meta_Equipment meta_equipment_obj;
        case EQUIPMENT_SD : Equipment equipment_obj;
        case LOCATION_TYPE_SD : Location_Type location_type_obj;
        case LOCATION_DESCRIPTION_SD : Location_Description location_descr_obj;
    };

    oneway void notify(
        in Domain_Repository::Repository_Item_Id id,
        in Static_Data_Object sd,
        in Domain_Repository::Change_Type change);
    oneway void abort();
};

#endif                                  /* __STATIC_DATA_OBSERVER_IDL__ */
```

The cleint interface through which notification data is delivered is as follow:

```
#ifndef __STATIC_DATA_OBSERVED_R_IDL__
#define __STATIC_DATA_OBSERVED_R_IDL__ 1

interface Static_Data_Observed_R {

    struct Observer_Entry {
        Static_Data_Observer obs;                 // Observer reference
        Repository_Item_Ids interest;             // observer interest
    };

    typedef sequence<Observer_Entry> Observer_Entries;
    readonly attribute Observer_Entries static_data_observer_list;

    readonly attribute Static_Data_Observed_E::Static_Data_Observers
            add_static_data_observer_list;
```

```
    void attach_static_data_observer(
        in Static_Data_Observer obs,
        in Repository_Item_Ids observed_id_list)
        raises(Unknown_Repository_Item_Ids);

    void detach_static_data_observer(
        in Static_Data_Observer obs,
        in Repository_Item_Ids observed_id_list)
        raises(Unknown_Repository_Item_Ids, Unknown_Observer,
               Unmatched_Observer_Item_Ids);

    void attach_add_static_data_observer(
        in Static_Data_Observer obs);

    void detach_add_static_data_observer(
        in Static_Data_Observer obs)
        raises(Domain_Repository::Unknown_Observer);
};

#endif                                  /* __STATIC_DATA_OBSERVED_R_IDL__ */

#ifndef __STATIC_DATA_OBSERVED_E_IDL__
#define __STATIC_DATA_OBSERVED_E_IDL__ 1

interface Static_Data_Observed_E {

    typedef sequence<Static_Data_Observer> Static_Data_Observers;
    readonly attribute Static_Data_Observers static_data_observer_list;

    void attach_static_data_observer(
        in Static_Data_Observer obs);

    void detach_static_data_observer(
        in Static_Data_Observer obs)
        raises(Unknown_Observer);
};

#endif                                  /* __STATIC_DATA_OBSERVED_E_IDL__ */
```

4.3 Location description

In order to describe localization of the any entities in the space, it has to be divided into sections. This partition could be arbitrary and is dependent only on a user. The description of these sections could be also discretional. Thus it was made possible for a user to define his list of properties for the element of space obtained using the given splitting.

4.3.1 Location Types Repository

This database stores types of different locations arrangement. These types are defined by specification of lists of properties whose types of these properties are expressed by CORBA concept of *TypeCode*.

```
#ifndef __LOCATION_TYPE_IDL__
#define __LOCATION_TYPE_IDL__ 1

interface Location_Type : Static_Data_Observed_E {

    typedef string Location_Type_Name;
    typedef String_Pattern Location_Type_Name_Pattern;

    struct Description {
        Repository_Item_Id item_id;
        Location_Type_Name name;
        Property_Descriptions property_description_list;
    };

    typedef sequence<Description> Descriptions;

    typedef Description Value_Description;
    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
        boolean is_any;
        Repository_Item_Id_Pattern item_id_pattern;
        Location_Type_Name_Pattern name_pattern;
        Property_Descriptions_Pattern
            property_description_list_pattern;
    };

    readonly attribute Description descr;
    readonly attribute Value_Description value_descr;
    readonly attribute Repository_Item_Id item_id;
    readonly attribute Location_Type_Name name;
    readonly attribute Property_Descriptions property_description_list;
};

#endif                                      /* __LOCATION_TYPE_IDL__ */
```

Object of this kind accessed via the repository interface from *Domain_Repository* module:

```
#ifndef __LOCATION_TYPE_REP_IDL__
#define __LOCATION_TYPE_REP_IDL__ 1

interface Location_Type_Rep : Static_Data_Observed_R {

    typedef sequence<Location_Type> Location_Types;

    Location_Types find(
        in Location_Type::Pattern data_record_pattern)
```

```
        raises(Search_Not_Implemented);
    Location_Type::Value_Descriptions find_values(
        in Location_Type::Pattern data_record_pattern)
        raises(Search_Not_Implemented);
};

#endif                                    /* __LOCATION_TYPE_REP_IDL__ */
```

and managed via the management interface from *Domain_Repository_Management* module:

```
#ifndef __LOCATION_TYPE_REP_MANAGER_IDL__
#define __LOCATION_TYPE_REP_MANAGER_IDL__ 1

interface Location_Type_Rep_Manager {

    readonly attribute Domain_Repository::Location_Type_Rep rep;
    readonly attribute Domain_Repository::Location_Type_Rep::Location_Types
        location_type_list;

    Domain_Repository::Location_Type add(
        in Domain_Repository::Location_Type::Description data_record)
        raises(Domain_Repository::Duplicate_Data_Record,
               Domain_Repository::Invalid_Description);
    void remove(in Domain_Repository::Location_Type object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref,
               Domain_Repository::Operation_Not_Allowed);
    void update(in Domain_Repository::Location_Type object_to_update,
        in Domain_Repository::Location_Type::Description data_record_pattern)
        raises(Domain_Repository::Unknown_Object_Ref,
               Domain_Repository::Duplicate_Data_Record,
               Domain_Repository::Invalid_Description);
};

#endif                          /* __LOCATION_TYPE_REP_MANAGER_IDL__ */
```

4.3.2 Location Descriptions Database

Using types of locations stored in the repository user could divide the given office environment into such parts. Each such type is described by list of properties with assigned to them appropriate values.

```
#ifndef __LOCATION_DESCRIPTION_IDL__
#define __LOCATION_DESCRIPTION_IDL__ 1

interface Location_Description : Static_Data_Observed_E {

    struct Description {
        Repository_Item_Id item_id;
        Location_Type::Location_Type_Name type_name;
        Properties property_list;
    };
```

```
    typedef sequence<Description> Descriptions;

    typedef Description Value_Description;
    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
        boolean is_any;
        Repository_Item_Id_Pattern item_id_pattern;
        Location_Type::Location_Type_Name_Pattern type_name_pattern;
        Properties_Pattern property_list_pattern;
    };

    readonly attribute Description descr;
    readonly attribute Value_Description value_descr;
    readonly attribute Repository_Item_Id item_id;

// Original:     readonly attribute Location_Type::Location_Type_Name type_name;
// typedef unwinded due to a bug in OrbixWeb 3.0
    readonly attribute string type_name;

    readonly attribute Properties property_list;
};

#endif                                        /* __LOCATION_DESCRIPTION_IDL__ */
```

Object of this kind accessed via the repository interface from *Domain_Repository* module:

```
#ifndef __LOCATION_DESCRIPTION_REP_IDL__
#define __LOCATION_DESCRIPTION_REP_IDL__ 1

interface Location_Description_Rep : Static_Data_Observed_R {

    typedef sequence<Location_Description> Location_Descriptions;

    Location_Descriptions find(
        in Location_Description::Pattern data_record_pattern)
        raises(Search_Not_Implemented);
    Location_Description::Value_Descriptions find_values(
        in Location_Description::Pattern data_record_pattern)
        raises(Search_Not_Implemented);
};

#endif                                    /* __LOCATION_DESCRIPTION_REP_IDL__ */
```

and managed via the management interface from *Domain_Repository_Management* module:

```
#ifndef __LOCATION_DESCRIPTION_REP_MANAGER_IDL__
#define __LOCATION_DESCRIPTION_REP_MANAGER_IDL__ 1

interface Location_Description_Rep_Manager {

    readonly attribute Domain_Repository::Location_Description_Rep rep;
    readonly attribute
```

```
        Domain_Repository::Location_Description_Rep::Location_Descriptions
                        location_description_list;

    Domain_Repository::Location_Description add(
        in Domain_Repository::Location_Description::Description data_record)
        raises(Domain_Repository::Duplicate_Data_Record,
            Domain_Repository::Invalid_Description);
    void remove(in Domain_Repository::Location_Description object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
    void update(in Domain_Repository::Location_Description object_to_update,
        in Domain_Repository::Location_Description::Description
                        data_record_pattern)
        raises(Domain_Repository::Unknown_Object_Ref,
            Domain_Repository::Duplicate_Data_Record,
            Domain_Repository::Invalid_Description);
};

#endif                          /* __LOCATION_DESCRIPTION_REP_MANAGER_IDL__ */
```

4.4 Locatable Databases

Locatables are entities localized by the Active Badge System. A lots of different properties could be associated which each of kind of them. At the moment there are two kinds of these entities: user and equipment, and standard sets of properties were however chosen for each of them. In the future dynamic definition of these set could be made possible.

4.4.1 Users Database

User is characterizes by the following attributes:

```
#ifndef __USER_IDL__
#define __USER_IDL__ 1

interface User : Static_Data_Observed_E {

    struct Description {
        Repository_Item_Id item_id;
        string uid;
        string forename;
        string surname;
        string position;
        string email;
        string phone;
    };

    typedef sequence<Description> Descriptions;

    typedef Description Value_Description;
    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
```

```
        boolean is_any;
        Repository_Item_Id_Pattern item_id_pattern;
        String_Pattern uid_pattern;
        String_Pattern forename_pattern;
        String_Pattern surname_pattern;
        String_Pattern position_pattern;
        String_Pattern email_pattern;
        String_Pattern phone_pattern;
    };

    readonly attribute Description descr;
    readonly attribute Value_Description value_descr;
    readonly attribute Repository_Item_Id item_id;
    readonly attribute string uid;
    readonly attribute string forename;
    readonly attribute string surname;
    readonly attribute string position;
    readonly attribute string email;
    readonly attribute string phone;
};

#endif                                              /* __USER_IDL__ */
```

uid is a unique User identification within the given ABng domain. It is assigned by the administrator during registration of the user within the system. Rest of the attributes are self explained.

Object of this kind accessed via the repository interface from *Domain_Repository* module:

```
#ifndef __USER_REP_IDL__
#define __USER_REP_IDL__ 1

interface User_Rep : Static_Data_Observed_R {

    typedef sequence<User> Users;

    Users find(in User::Pattern data_record_pattern)
        raises(Search_Not_Implemented);
    User::Value_Descriptions find_values(
        in User::Pattern data_record_pattern)
        raises(Search_Not_Implemented);
};

#endif                                          /* __USER_REP_IDL__ */
```

and managed via the management interface from *Domain_Repository_Management* module:

```
#ifndef __USER_REP_MANAGER_IDL__
#define __USER_REP_MANAGER_IDL__ 1

interface User_Rep_Manager {

    readonly attribute Domain_Repository::User_Rep rep;
    readonly attribute Domain_Repository::User_Rep::Users user_list;
```

```
Domain_Repository::User add(
    in Domain_Repository::User::Description data_record)
    raises(Domain_Repository::Duplicate_Data_Record);

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

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

#endif                                        /* __USER_REP_MANAGER_IDL__ */
```

4.4.2 MetaEquipments and Equipments Database

Equipment installed inside the building could be divided into different categories. These categories are stored as a C++ objects (made persistent using ObjectStore) inside MetaEquipment Database. Each category of equipment is describe by the following properties:

Meta_type is a structure consisting of a bitmap, which will represent this category of equipment in the visualization tools and name of the given category. The name is taken from the vocabulary created by the system administrator. It consists of typical equipment names: *TV-set*, *VCR*, *Microwave*, *Air-conditioner*, etc. *vendor* and *model* are self explaining. *my_dimensions* contains rough dimensions of this category of the equipment. The two last properties are related to Infra-red Control. Some categories of equipment could be controlled using infra-red. For these categories *IR_controled* attribute is set to *true* and *my_IR_control_data* contains information necessary to invoke infra-red commands on the equipment:

```
#ifndef __IRTYPES_IDL__
#define __IRTYPES_IDL__ 1

typedef Strings Command_List;
typedef Strings_Pattern Command_List_Pattern;
typedef Octets IR_Data;
typedef Octets_Pattern IR_Data_Pattern;

struct IR_Control_Data {
    Command_List commands;
    IR_Data my_IR_data;
};

struct IR_Control_Data_Pattern {
    boolean is_any;
    Command_List_Pattern commands_pattern;
    IR_Data_Pattern my_IR_data_pattern;
};
```

```
#endif                                              /* __IRTYPES_IDL__ */
```

This data contains list of available commands and infra-red sequences corresponding to them.

Real instances of different categories of equipment are represented as a CORBA object. Data from the MetaEquipment database is replicated in them due to the performance reason. Additionally equipment posses unique *name*. There is also recorded name of the last_command, which was lastly invoked on it, obviously if this equipment is infra-red controlled.

```
#ifndef __EQUIPMENT_IDL__
#define __EQUIPMENT_IDL__ 1

interface Equipment : Static_Data_Observed_E {

    struct Description {
        Repository_Item_Id item_id;
        Meta_Equipment meta_data;
        string name;
        string last_command;
        Properties property_list;
    };

    typedef sequence<Description> Descriptions;

    struct Value_Description {
        Repository_Item_Id item_id;
        Meta_Equipment::Description meta_data;
        string name;
        string last_command;
        Properties property_list;
    };
    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
        boolean is_any;
        Repository_Item_Id_Pattern item_id_pattern;
        Meta_Equipment::Pattern meta_data_pattern;
        String_Pattern name_pattern;
        String_Pattern last_command_pattern;
        Properties_Pattern property_list_pattern;
    };

   Description descr()
        raises(Destination_Unreachable);

    Value_Description value_descr()
        raises(Destination_Unreachable);

    readonly attribute Repository_Item_Id item_id;

    Meta_Equipment meta_data()
        raises(Destination_Unreachable);
```

```
    readonly attribute string name;
    readonly attribute string last_command;
    readonly attribute Properties property_list;

};

#endif                                          /* __EQUIPMENT_IDL__ */
```

Object of this kind accessed via the repository interface from *Domain_Repository* module:

```
#ifndef __EQUIPMENT_REP_IDL__
#define __EQUIPMENT_REP_IDL__ 1

interface Equipment_Rep : Static_Data_Observed_R {

    typedef sequence<Equipment> Equipments;

    Equipments find(in Equipment::Pattern data_record_pattern)
        raises(Search_Not_Implemented,
               Destination_Unreachable);
    Equipment::Value_Descriptions find_values(
        in Equipment::Pattern data_record_pattern)
        raises(Search_Not_Implemented,
               Destination_Unreachable);
};

#endif                                          /* __EQUIPMENT_REP_IDL__ */
```

and managed via the management interface from *Domain_Repository_Management* module:

```
#ifndef __EQUIPMENT_REP_MANAGER_IDL
#define __EQUIPMENT_REP_MANAGER_IDL 1

interface Equipment_Rep_Manager {

    readonly attribute Domain_Repository::Equipment_Rep rep;
    readonly attribute Domain_Repository::Equipment_Rep::Equipments
            equipment_list;

    Domain_Repository::Equipment add(in
        Domain_Repository::Equipment::Description data_record)
        raises(Domain_Repository::Duplicate_Data_Record,
               Domain_Repository::Invalid_Description,
               Domain_Repository::Destination_Unreachable);
    void remove(in Domain_Repository::Equipment object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
    void update(in Domain_Repository::Equipment object_to_update,
        in Domain_Repository::Equipment::Description data_record_pattern)
        raises(Domain_Repository::Unknown_Object_Ref,
               Domain_Repository::Duplicate_Data_Record,
               Domain_Repository::Invalid_Description,
               Domain_Repository::Destination_Unreachable);
```

```
};

#endif                                         /* __EQUIPMENT_REP_MANAGER_IDL */
```

5 ABng Interface Description

The ABng interfaces have been grouped into seven IDL modules:

- *ABng_Core* – API for ABng clients.
- *ABng_Poller* – poller and poller observer interfaces.
- *ABng_Core_System* – interfaces for objects representing ABng system components, such as badges, sensors, the scheduler, badge holders, etc.
- *ABng_Core_Internals* – interfaces of sighting processors used within the core internally for sighting processing.
- *ABng_Core_Management* – management interfaces.
- *Domain_Repository* – API of universal databases (e.g. user, equipment, location description database).
- *Domain_Repository_Management* – management interfaces of universal databases.

5.1 ABng API

The *ABng_Core* module is defined as follows.

```
#include <orb.idl>

#include "domain_rep.idl"

module ABng_Core {

interface Dynamic_Data_Observer ;
interface Dynamic_Data_Observed_E ;
interface Dynamic_Data_Observed_R ;
interface Request_Queue ;
interface Commander ;
interface Locatable_Commander ;
interface Location_Commander ;
interface Locatable ;
interface Location ;
interface Caller ;
interface View_Data_Observed_E ;
interface View_Data_Observed_R ;
interface View_Data_Observer ;
interface View ;
interface View_Rep ;
```

```
interface ABng_Domain ;

typedef Domain_Repository :: Octets Octets ;
typedef Domain_Repository :: Octets_Pattern Octets_Pattern ;
typedef Domain_Repository :: Time Time ;

typedef octet Seq_Number ;
typedef unsigned short Quality ;

enum Locatable_Type {
LOCATABLE_USER ,
LOCATABLE_EQUIPMENT
} ;

struct Sighting_Description {
string view_name ;
Domain_Repository :: Repository_Item_Id my_locatable ;
Locatable_Type my_locatable_type ;
Domain_Repository :: Repository_Item_Id my_location ;
octet message_type ;
octet attributes ;
Seq_Number sequence_number ;
Time update_time ;
Quality q ;
} ;

enum Command_Error_Code { SIGHTING_TIMEOUT , REPLY_TIMEOUT ,
POLLER_UNREACHABLE } ;




const unsigned short CLICK_NUMBER_MASK = 0xf ;
const unsigned short STATE_MASK = 0x30 ;

typedef sequence < Sighting_Description > Sighting_Descriptions ;

union Locatable_Type_Specific_Object switch ( Locatable_Type ) {
case LOCATABLE_USER : Domain_Repository :: User user_obj ;
case LOCATABLE_EQUIPMENT : Domain_Repository :: Equipment equipment_obj ;
} ;

struct Locatable_Type_Specific_Object_Pattern {
boolean is_any ;
union Ref_Pattern switch ( Locatable_Type ) {
case LOCATABLE_USER : Domain_Repository :: User :: Pattern
user_obj_pattern ;
case LOCATABLE_EQUIPMENT : Domain_Repository :: Equipment :: Pattern
equipment_obj_pattern ;
} object_pattern ;
```

```
} ;

union Locatable_Type_Specific_Object_Value switch ( Locatable_Type ) {
case LOCATABLE_USER : Domain_Repository :: User :: Value_Description
user_description ;
case LOCATABLE_EQUIPMENT : Domain_Repository :: Equipment :: Value_Description
equipment_description ;
} ;

enum Caller_Interest {
Error ,
Reply ,
Request_Invoked ,
Error_And_Reply ,
Error_And_Request_Invoked ,
None
} ;


struct Command_Context {
Time at ;
Time waiting_for_sighting_timeout ;
Time waiting_for_reply_timeout ;
unsigned short retries ;
Time interval ;
unsigned short repeat ;
Caller_Interest interest ;
} ;

typedef sequence < Location > Locations ;
typedef sequence < Locatable > Locatables ;

exception Invalid_Request_Time { } ;
exception Invalid_Argument { } ;
exception Unknown_Request_Id { } ;
exception Unknown_View_Name { } ;

interface Request_Queue {

struct Request {
string uid ;
long request_id ;
Locations location_list ;
Locatables locatable_list ;
Octets arguments ;
string command_name ;
Command_Context command_context ;
Time request_time ;
} ;

typedef sequence < Request > Requests ;
```

```
Requests list_user_requests ( in string user_id )
raises ( Domain_Repository :: Destination_Unreachable ) ;
Requests list_all_requests ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
void remove_request ( in long request_id )
raises ( Unknown_Request_Id ,
Domain_Repository :: Destination_Unreachable ) ;
} ;

interface Caller {

oneway void challenge_reply (
in Location my_location ,
in Locatable my_locatable ,
in Octets random ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void sound_reply (
in Location my_location ,
in Locatable my_locatable ,
in Octets sound ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void set_state_reply (
in Location my_location ,
in Locatable my_locatable ,
in octet status ,
in Octets sound ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void get_state_reply (
in Location my_location ,
in Locatable my_locatable ,
in octet major_version ,
in octet minor_version ,
in octet status ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void set_home_reply (
in Location my_location ,
in Locatable my_locatable ,
in Octets address ,
in Time ts ,
in long request_id ,
```

```
in long repeat_num ) ;

oneway void get_home_reply (
in Location my_location ,
in Locatable my_locatable ,
in Octets address ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void get_station_state_reply (
in Location my_location ,
in octet major_version ,
in octet minor_version ,
in octet status1 ,
in octet status2 ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void location_command_reply (
in Location my_location ,
in Octets reply_body ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void locatable_command_reply (
in Location my_location ,
in Locatable my_locatable ,
in Octets reply_body ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void null_message (
in Location my_location ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void request_processed (
in Location my_location ,
in Locatable my_locatable ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void error_occured (
in Location my_location ,
in Locatable my_locatable ,
in ABng_Core :: Command_Error_Code error_code ,
```

```
in long request_id ,
in long repeat_num ,
in long retry_num ) ;

oneway void request_completed (
in long request_id ) ;

} ;

interface Commander : Request_Queue {

long issue_location_command (
in Locations location_list ,
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long issue_locatable_command (
in Locations location_list ,
in Locatables locatable_list ,
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long challenge (
in Locations location_list ,
in Locatables locatable_list ,
in Octets random ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long sound (
in Locations location_list ,
in Locatables locatable_list ,
in Octets sound_ ,
in Command_Context ctx ,
```

```
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_state (
in Locations location_list ,
in Locatables locatable_list ,
in octet status ,
in Octets sound_ ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_state (
in Locations location_list ,
in Locatables locatable_list ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_home (
in Locations location_list ,
in Locatables locatable_list ,
in Octets address ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_home (
in Locations location_list ,
in Locatables locatable_list ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long reset_buffer (
in Locations location_list ,
in Command_Context ctx ,
```

```
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long reset_fifos (
in Locations location_list ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_station_state (
in Locations location_list ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_output (
in Locations location_list ,
in octet output_value ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_persistence (
in Locations location_list ,
in boolean mode ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;
} ;

interface Locatable_Commander : Request_Queue {

long issue_command (
in Locations location_list ,
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
```

```
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long challenge (
in Locations location_list ,
in Octets random ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long sound (
in Locations location_list ,
in Octets sound_ ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_state (
in Locations location_list ,
in octet status ,
in Octets sound_ ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_state (
in Locations location_list ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_home (
in Locations location_list ,
in Octets address ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_home (
```

```
in Locations location_list ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;
} ;

interface Location_Commander : Request_Queue {

long issue_location_command (
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long issue_locatable_command (
in Locatables locatable_list ,
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long challenge (
in Locatables locatable_list ,
in Octets random ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long sound (
in Locatables locatable_list ,
in Octets sound_ ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_state (
in Locatables locatable_list ,
in octet status ,
```

```
in Octets sound_ ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_state (
in Locatables locatable_list ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_home (
in Locatables locatable_list ,
in Octets address ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_home (
in Locatables locatable_list ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;


long reset_buffer (
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long reset_fifos (
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_station_state (
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
```

```
Domain_Repository :: Destination_Unreachable ) ;

long set_output (
in octet output_value ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_persistence (
in boolean mode ,
in Command_Context ctx ,
in Caller reply_to )
raises ( Invalid_Request_Time ,
Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

} ;

interface Dynamic_Data_Observed_E {

typedef sequence < Dynamic_Data_Observer > Dynamic_Data_Observers ;
readonly attribute Dynamic_Data_Observers dynamic_data_observer_list ;

void attach_dynamic_data_observer (
in Dynamic_Data_Observer obs ) ;

void detach_dynamic_data_observer (
in Dynamic_Data_Observer obs )
raises ( Domain_Repository :: Unknown_Observer ) ;
} ;

interface Dynamic_Data_Observed_R {

struct Observer_Entry {
Dynamic_Data_Observer obs ;
Domain_Repository :: Repository_Item_Ids interest ;
} ;

typedef sequence < Observer_Entry > Observer_Entries ;
readonly attribute Observer_Entries
location_dynamic_data_observer_list ;
readonly attribute Observer_Entries
locatable_dynamic_data_observer_list ;

void attach_location_dynamic_data_observer (
in Dynamic_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ) ;

void detach_location_dynamic_data_observer (
```

```
in Dynamic_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ,
Domain_Repository :: Unknown_Observer ,
Domain_Repository :: Unmatched_Observer_Item_Ids ) ;

void attach_locatable_dynamic_data_observer (
in Dynamic_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ) ;

void detach_locatable_dynamic_data_observer (
in Dynamic_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ,
Domain_Repository :: Unknown_Observer ,
Domain_Repository :: Unmatched_Observer_Item_Ids ) ;
} ;

interface Dynamic_Data_Observer {
oneway void receive_sighting ( in Sighting_Description sd ) ;

oneway void abort ( ) ;
} ;

interface View_Data_Observed_E {

typedef sequence < View_Data_Observer > View_Data_Observers ;
readonly attribute View_Data_Observers view_data_observer_list ;

void attach_view_data_observer (
in View_Data_Observer obs ) ;

void detach_view_data_observer (
in View_Data_Observer obs )
raises ( Domain_Repository :: Unknown_Observer ) ;
} ;

interface View_Data_Observed_R {

struct Observer_Entry {
View_Data_Observer obs ;
Domain_Repository :: Repository_Item_Ids interest ;
} ;

typedef sequence < Observer_Entry > Observer_Entries ;
readonly attribute Observer_Entries location_view_data_observer_list ;
readonly attribute View_Data_Observed_E :: View_Data_Observers
location_add_view_data_observer_list ;

readonly attribute Observer_Entries locatable_view_data_observer_list ;
readonly attribute View_Data_Observed_E :: View_Data_Observers
```

```
locatable_add_view_data_observer_list ;

void attach_location_view_data_observer (
in View_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ) ;

void detach_location_view_data_observer (
in View_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ,
Domain_Repository :: Unknown_Observer ,
Domain_Repository :: Unmatched_Observer_Item_Ids ) ;

void attach_location_add_view_data_observer (
in View_Data_Observer obs ) ;

void detach_location_add_view_data_observer (
in View_Data_Observer obs )
raises ( Domain_Repository :: Unknown_Observer ) ;

void attach_locatable_view_data_observer (
in View_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ) ;

void detach_locatable_view_data_observer (
in View_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ,
Domain_Repository :: Unknown_Observer ,
Domain_Repository :: Unmatched_Observer_Item_Ids ) ;

void attach_locatable_add_view_data_observer (
in View_Data_Observer obs ) ;

void detach_locatable_add_view_data_observer (
in View_Data_Observer obs )
raises ( Domain_Repository :: Unknown_Observer ) ;
} ;

interface View_Data_Observer {

enum View_Data_Type {
LOCATION_VD ,
LOCATABLE_VD
} ;

union View_Data_Object switch ( View_Data_Type ) {
case LOCATION_VD : Location location_obj ;
case LOCATABLE_VD : Locatable locatable_obj ;
} ;
```

```
oneway void notify (
in Domain_Repository :: Repository_Item_Id id ,
in View_Data_Object sd ,
in Domain_Repository :: Change_Type change ) ;
oneway void abort ( ) ;
} ;

interface Location : Dynamic_Data_Observed_E ,
View_Data_Observed_E ,
Location_Commander {

struct Description {
Domain_Repository :: Repository_Item_Id item_id ;
Domain_Repository :: Location_Description loc_descr ;
} ;

typedef sequence < Description > Descriptions ;

struct Value_Description {
Domain_Repository :: Repository_Item_Id item_id ;
Domain_Repository :: Location_Description :: Value_Description
loc_value_descr ;
} ;

typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
Domain_Repository :: Repository_Item_Id_Pattern item_id_pattern ;
Domain_Repository :: Location_Description :: Pattern loc_descr_pattern ;
} ;

readonly attribute Domain_Repository :: Repository_Item_Id item_id ;

Description descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;

Value_Description value_descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;

Domain_Repository :: Location_Description loc_descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;

Sighting_Descriptions get_sightings ( in Time since ) ;
} ;

interface Locatable : Dynamic_Data_Observed_E ,
View_Data_Observed_E ,
Locatable_Commander {

struct Description {
```

```
Domain_Repository :: Repository_Item_Id item_id ;
Location home_location ;
Locatable_Type_Specific_Object type_specific_object ;
} ;

typedef sequence < Description > Descriptions ;

struct Value_Description {
Domain_Repository :: Repository_Item_Id item_id ;
Location :: Value_Description home_location ;
Locatable_Type_Specific_Object_Value
type_specific_object_value ;
} ;

typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
Domain_Repository :: Repository_Item_Id_Pattern item_id_pattern ;
Location :: Pattern home_location_pattern ;
Locatable_Type_Specific_Object_Pattern
type_specific_object_pattern ;
} ;

readonly attribute Domain_Repository :: Repository_Item_Id item_id ;

Description descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;

Value_Description value_descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;

Location home_location ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;


Locatable_Type_Specific_Object type_specific_object ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;

Sighting_Description get_sighting ( ) ;
} ;

interface View : Dynamic_Data_Observed_R ,
View_Data_Observed_R ,
Commander {

readonly attribute string view_name ;

readonly attribute Domain_Repository :: Location_Type :: Location_Type_Name type ;
readonly attribute Locations location_list ;
readonly attribute Locatables locatable_list ;
readonly attribute boolean wants_duplicates ;
```

```
Locations find_location (
in Location :: Pattern data_record_pattern )
raises ( Domain_Repository :: Search_Not_Implemented ,
Domain_Repository :: Destination_Unreachable ) ;
Location :: Value_Descriptions find_location_values (
in Location :: Pattern data_record_pattern )
raises ( Domain_Repository :: Search_Not_Implemented ,
Domain_Repository :: Destination_Unreachable ) ;

Locatables find_locatable (
in Locatable :: Pattern data_record_pattern )
raises ( Domain_Repository :: Search_Not_Implemented ,
Domain_Repository :: Destination_Unreachable ) ;
Locatable :: Value_Descriptions find_locatable_values (
in Locatable :: Pattern data_record_pattern )
raises ( Domain_Repository :: Search_Not_Implemented ,
Domain_Repository :: Destination_Unreachable ) ;

Sighting_Descriptions get_sightings ( in Time since ) ;

} ;

interface View_Rep {

typedef sequence < View > Views ;

Views find ( in string name )
raises ( Domain_Repository :: Search_Not_Implemented ) ;

Views get_all ( ) ;
} ;

interface ABng_Domain {

readonly attribute Domain_Repository :: Location_Description
domain_description ;
readonly attribute Domain_Repository :: Local_Dependence
domain_local_dependence ;
} ;

} ;
```

The IDL files included in the above definition are presented and described below. The *abng_core_types.idl* file contains common type definitions including the *Sighting_Description* structure used for sending sighting data to applications and the *Command_Context* structure used to specify constraints for commands sent to locations and locatables.

```
#ifndef __ABNG_CORE_TYPES_IDL__
#define __ABNG_CORE_TYPES_IDL__ 1

interface Dynamic_Data_Observer;
```

```
interface Dynamic_Data_Observed_E;
interface Dynamic_Data_Observed_R;
interface Request_Queue;
interface Commander;
interface Locatable_Commander;
interface Location_Commander;
interface Locatable;
interface Location;
interface Caller;
interface View_Data_Observed_E;
interface View_Data_Observed_R;
interface View_Data_Observer;
interface View;
interface View_Rep;
interface ABng_Domain;

// Common type definitions
typedef Domain_Repository::Octets Octets;
typedef Domain_Repository::Octets_Pattern Octets_Pattern;
typedef Domain_Repository::Time Time;

typedef octet Seq_Number;
typedef unsigned short Quality;

enum Locatable_Type {
    LOCATABLE_USER,
    LOCATABLE_EQUIPMENT
};

struct Sighting_Description {
    string view_name;
    Domain_Repository::Repository_Item_Id my_locatable;
    Locatable_Type my_locatable_type;
    Domain_Repository::Repository_Item_Id my_location;
    octet message_type;
    octet attributes;
    Seq_Number sequence_number;
    Time update_time;
    Quality q;
};

enum Command_Error_Code { SIGHTING_TIMEOUT, REPLY_TIMEOUT,
                          POLLER_UNREACHABLE };
// attributes have the following format: xxxabbbb, where:
//        <a> - the state of the badge output (for equipment only)
//        <bbbb> - number of clicks.
// To obtain these data the following bit masks can be used:

const unsigned short CLICK_NUMBER_MASK = 0xf;
const unsigned short STATE_MASK = 0x30;

typedef sequence<Sighting_Description> Sighting_Descriptions;
```

```
union Locatable_Type_Specific_Object switch(Locatable_Type) {
    case LOCATABLE_USER: Domain_Repository::User user_obj;
    case LOCATABLE_EQUIPMENT: Domain_Repository::Equipment equipment_obj;
};

struct Locatable_Type_Specific_Object_Pattern {
    boolean is_any;
    union Ref_Pattern switch(Locatable_Type) {
        case LOCATABLE_USER: Domain_Repository::User::Pattern
            user_obj_pattern;
        case LOCATABLE_EQUIPMENT: Domain_Repository::Equipment::Pattern
            equipment_obj_pattern;
    } object_pattern;
};

union Locatable_Type_Specific_Object_Value switch(Locatable_Type) {
    case LOCATABLE_USER: Domain_Repository::User::Value_Description
        user_description;
    case LOCATABLE_EQUIPMENT: Domain_Repository::Equipment::Value_Description
        equipment_description;
};

enum Caller_Interest {
    Error,                          // Inform about an error
    Reply,                          // Send reply
    Request_Invoked,                // Call back after receiveing a reply
    Error_And_Reply,                // Inform about an error or send a reply
    Error_And_Request_Invoked,      // Inform about an error or request invocation
    None                            // Never call back
};

// Constraints for command execution
struct Command_Context {
    Time at;                                // Time when a request execution is started.
    Time waiting_for_sighting_timeout;      //
    Time waiting_for_reply_timeout;         // Timeout for a single try
    unsigned short retries;                 // A number of tries
    Time interval;                          // Interval between two command executions.
    unsigned short repeat;                  // Number of command executions
    Caller_Interest interest;               // When a caller should be called back
};

typedef sequence<Location> Locations;
typedef sequence<Locatable> Locatables;

exception Invalid_Request_Time {};
exception Invalid_Argument {};
exception Unknown_Request_Id {};
exception Unknown_View_Name {};

#endif                                  /* __ABNG_CORE_TYPES_IDL__ */
```

Interfaces defined in the next five included files are used by ABng clients for sending commands to locatables and locations.

The *Request_Queue* base interface contains operations used to manage the command queue. *Caller* is a call-back interface which is implemented by an application and is invoked by ABng to send command replies, to report command errors or to inform an application about command completion. When an application invokes a command it can specify in the context parameter which of these three messages are to be sent.

Commander is the base interface for *View* and is used to send commands to location and locatable objects that are contained within a view. Particular method names correspond to commands accepted by locations (sensors installed within a given location) or locatables (badges worn by users and equipment). The first two methods do not represent any command. They may be used to send any message to a location or a locatable. Two first parameters of each method that send a command to locations is a list of locations. An empty list means all locations within a given view. The remaining parameters are command-specific attributes, context and a caller reference (it may be the *nil* reference, if an application does not want to receive a reply, error and success report). The method causes the given command to be executed in all locations (in all sensors in all locations) from the list.

Methods responsible for sending commands to locatables take an additional parameter which is a list locatables to which a command is sent. An empty list means all locatables in the view. The meaning of the location list is different: it specifies locations through which a command may be sent to the given locatable objects. If the location list is empty a command will be sent to a locatable object through the current object location. If a locatable is not visible in any specified location (or is not visible at all if the location list is empty) a command will wait (will be queued by the underlying scheduler) until the locatable object turns up. The maximum waiting time may be specified in the context parameter.

```
#ifndef __REQUEST_QUEUE_IDL__
#define __REQUEST_QUEUE_IDL__ 1

interface Request_Queue {

    struct Request {
        string uid;
        long request_id;
        Locations location_list;
        Locatables locatable_list;
        Octets arguments;
        string command_name;
        Command_Context command_context;
        Time request_time;
    };

    typedef sequence<Request> Requests;

    Requests list_user_requests(in string user_id)
        raises (Domain_Repository::Destination_Unreachable);
    Requests list_all_requests()
        raises (Domain_Repository::Destination_Unreachable);
```

```
    void remove_request(in long request_id)
        raises (Unknown_Request_Id,
            Domain_Repository::Destination_Unreachable);
};

#endif                                          /* __REQUEST_QUEUE_IDL__ */

#ifndef __CALLER_IDL__
#define __CALLER_IDL__ 1

// The interface used for sending command replies. Should be provided
// by a command caller.

interface Caller {

    oneway void challenge_reply(
        in Location my_location,
        in Locatable my_locatable,
        in Octets random,
        in Time ts,
        in long request_id,
        in long repeat_num);

    oneway void sound_reply(
        in Location my_location,
        in Locatable my_locatable,
        in Octets sound,
        in Time ts,
        in long request_id,
        in long repeat_num);

    oneway void set_state_reply(
        in Location my_location,
        in Locatable my_locatable,
        in octet status,
        in Octets sound,
        in Time ts,
        in long request_id,
        in long repeat_num);

    oneway void get_state_reply(
        in Location my_location,
        in Locatable my_locatable,
        in octet major_version,
        in octet minor_version,
        in octet status,
        in Time ts,
        in long request_id,
        in long repeat_num);

    oneway void set_home_reply(
        in Location my_location,
        in Locatable my_locatable,
```

```
        in Octets address,
        in Time ts,
        in long request_id,
        in long repeat_num);

oneway void get_home_reply(
        in Location my_location,
        in Locatable my_locatable,
        in Octets address,
        in Time ts,
        in long request_id,
        in long repeat_num);

oneway void get_station_state_reply(
        in Location my_location,
        in octet major_version,
        in octet minor_version,
        in octet status1,
        in octet status2,
        in Time ts,
        in long request_id,
        in long repeat_num);

oneway void location_command_reply(
        in Location my_location,
        in Octets reply_body,
        in Time ts,
        in long request_id,
        in long repeat_num);

oneway void locatable_command_reply(
        in Location my_location,
        in Locatable my_locatable,
        in Octets reply_body,
        in Time ts,
        in long request_id,
        in long repeat_num);

oneway void null_message(
        in Location my_location,
        in Time ts,
        in long request_id,
        in long repeat_num);

oneway void request_processed(
        in Location my_location,
        in Locatable my_locatable,
        in Time ts,
        in long request_id,
        in long repeat_num);

oneway void error_occured(
```

```
        in Location my_location,
        in Locatable my_locatable,
        in ABng_Core::Command_Error_Code error_code,
        in long request_id,
        in long repeat_num,
        in long retry_num);

    oneway void request_completed(
        in long request_id);

};

#endif                                                    /* __CALLER_IDL__ */

#ifndef __COMMANDER_IDL__
#define __COMMANDER_IDL__ 1

interface Commander : Request_Queue {

    long issue_location_command(
        in Locations location_list,
        in Octets cmd,
        in boolean has_reply,
        in octet reply_code,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Domain_Repository::Unknown_Object_Ref,
            Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long issue_locatable_command(
        in Locations location_list,
        in Locatables locatable_list,
        in Octets cmd,
        in boolean has_reply,
        in octet reply_code,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Domain_Repository::Unknown_Object_Ref,
            Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long challenge(
        in Locations location_list,
        in Locatables locatable_list,
        in Octets random,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Domain_Repository::Unknown_Object_Ref,
            Invalid_Request_Time,
            Invalid_Argument,
```

```
        Domain_Repository::Destination_Unreachable);

long sound(
    in Locations location_list,
    in Locatables locatable_list,
    in Octets sound_,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Domain_Repository::Unknown_Object_Ref,
        Invalid_Request_Time,
        Invalid_Argument,
        Domain_Repository::Destination_Unreachable);

long set_state(
    in Locations location_list,
    in Locatables locatable_list,
    in octet status,
    in Octets sound_,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Domain_Repository::Unknown_Object_Ref,
        Invalid_Request_Time,
        Invalid_Argument,
        Domain_Repository::Destination_Unreachable);

long get_state(
    in Locations location_list,
    in Locatables locatable_list,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Domain_Repository::Unknown_Object_Ref,
        Invalid_Request_Time,
        Invalid_Argument,
        Domain_Repository::Destination_Unreachable);

long set_home(
    in Locations location_list,
    in Locatables locatable_list,
    in Octets address,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Domain_Repository::Unknown_Object_Ref,
        Invalid_Request_Time,
        Invalid_Argument,
        Domain_Repository::Destination_Unreachable);

long get_home(
    in Locations location_list,
    in Locatables locatable_list,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Domain_Repository::Unknown_Object_Ref,
```

```
            Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long reset_buffer(
        in Locations location_list,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Domain_Repository::Unknown_Object_Ref,
            Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long reset_fifos(
        in Locations location_list,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Domain_Repository::Unknown_Object_Ref,
            Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long get_station_state(
        in Locations location_list,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Domain_Repository::Unknown_Object_Ref,
            Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long set_output(
        in Locations location_list,
        in octet output_value,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Domain_Repository::Unknown_Object_Ref,
            Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long set_persistence(
        in Locations location_list,
        in boolean mode,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Domain_Repository::Unknown_Object_Ref,
            Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);
};
```

```
#endif                                        /* __COMMANDER_IDL__ */
```

For a programmer's convenience, the methods that send location and locatables commands have been separated and grouped into the *Location_Commander* and *Locatable_Commander* interfaces respectively. They are the base interfaces for the *Location* and *Locatable* objects described below. The CORBA object, for which methods of *Location_Commander* or *Locatable_Commander* are invoked, determines the target command location or locatable. Therefore, the location list and locatable list parameters are not needed any longer in methods of *Location_Commander* or *Locatable_Commander* respectively and have been removed from their definitions.

Implementations of the *Commander*, *Location_Commander* and *Locatable_Commander* interfaces convert the obtained list of location and locatable objects into lists of sensors and badges, reformat method parameters (if necessary), and call the lower level system interfaces: *Scheduler*, *Sensor_Commander* or *Badge_Commander*.

All three commander interfaces are derived from the *Request_Queue*. This does not mean that each view, location or locatable object implements its own command queue. In fact, each of the three implementations of that interface operates on the queue managed by the scheduler.

```
#ifndef __LOCATABLE_COMMANDER_IDL__
#define __LOCATABLE_COMMANDER_IDL__ 1

interface Locatable_Commander : Request_Queue {

    long issue_command(
        in Locations location_list,
        in Octets cmd,
        in boolean has_reply,
        in octet reply_code,
        in Command_Context ctx,
        in Caller reply_to)
        raises(Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long challenge(
        in Locations location_list,
        in Octets random,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long sound(
        in Locations location_list,
        in Octets sound_,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
```

```
            Domain_Repository::Destination_Unreachable);

    long set_state(
        in Locations location_list,
        in octet status,
        in Octets sound_,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long get_state(
        in Locations location_list,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long set_home(
        in Locations location_list,
        in Octets address,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long get_home(
        in Locations location_list,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);
};

#endif                                    /* __LOCATABLE_COMMANDER_IDL__ */

#ifndef __LOCATION_COMMANDER_IDL__
#define __LOCATION_COMMANDER_IDL__ 1

interface Location_Commander : Request_Queue {

    long issue_location_command(
        in Octets cmd,
        in boolean has_reply,
        in octet reply_code,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
```

```
        Domain_Repository::Destination_Unreachable);

long issue_locatable_command(
    in Locatables locatable_list,
    in Octets cmd,
    in boolean has_reply,
    in octet reply_code,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Invalid_Request_Time,
        Invalid_Argument,
        Domain_Repository::Destination_Unreachable);

long challenge(
    in Locatables locatable_list,
    in Octets random,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Invalid_Request_Time,
        Invalid_Argument,
        Domain_Repository::Destination_Unreachable);

long sound(
    in Locatables locatable_list,
    in Octets sound_,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Invalid_Request_Time,
        Invalid_Argument,
        Domain_Repository::Destination_Unreachable);

long set_state(
    in Locatables locatable_list,
    in octet status,
    in Octets sound_,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Invalid_Request_Time,
        Invalid_Argument,
        Domain_Repository::Destination_Unreachable);

long get_state(
    in Locatables locatable_list,
    in Command_Context ctx,
    in Caller reply_to)
    raises (Invalid_Request_Time,
        Invalid_Argument,
        Domain_Repository::Destination_Unreachable);

long set_home(
    in Locatables locatable_list,
    in Octets address,
```

```
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long get_home(
        in Locatables locatable_list,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    // Sensor commands
    long reset_buffer(
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long reset_fifos(
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long get_station_state(
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long set_output(
        in octet output_value,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long set_persistence(
        in boolean mode,
        in Command_Context ctx,
        in Caller reply_to)
        raises (Invalid_Request_Time,
            Invalid_Argument,
            Domain_Repository::Destination_Unreachable);
```

```
};

#endif                                   /* __LOCATION_COMMANDER_IDL__ */
```

The next three base interfaces define the way in which dynamic location data are provided to the application layer. This method is based on the orbserved/observer design pattern.

```
#ifndef __DYNAMIC_DATA_OBSERVED_R_IDL__
#define __DYNAMIC_DATA_OBSERVED_R_IDL__ 1

interface Dynamic_Data_Observed_R {

    struct Observer_Entry {
        Dynamic_Data_Observer obs;                              // observer reference
        Domain_Repository::Repository_Item_Ids interest;        // observer interest
    };

    typedef sequence<Observer_Entry> Observer_Entries;
    readonly attribute Observer_Entries
        location_dynamic_data_observer_list;
    readonly attribute Observer_Entries
        locatable_dynamic_data_observer_list;

    void attach_location_dynamic_data_observer(
        in Dynamic_Data_Observer obs,
        in Domain_Repository::Repository_Item_Ids observed_id_list)
        raises(Domain_Repository::Unknown_Repository_Item_Ids);

    void detach_location_dynamic_data_observer(
        in Dynamic_Data_Observer obs,
        in Domain_Repository::Repository_Item_Ids observed_id_list)
        raises(Domain_Repository::Unknown_Repository_Item_Ids,
               Domain_Repository::Unknown_Observer,
               Domain_Repository::Unmatched_Observer_Item_Ids);

    void attach_locatable_dynamic_data_observer(
        in Dynamic_Data_Observer obs,
        in Domain_Repository::Repository_Item_Ids observed_id_list)
        raises(Domain_Repository::Unknown_Repository_Item_Ids);

    void detach_locatable_dynamic_data_observer(
        in Dynamic_Data_Observer obs,
        in Domain_Repository::Repository_Item_Ids observed_id_list)
        raises(Domain_Repository::Unknown_Repository_Item_Ids,
               Domain_Repository::Unknown_Observer,
               Domain_Repository::Unmatched_Observer_Item_Ids);
};

#endif                                   /* __DYNAMIC_DATA_OBSERVED_R_IDL__ */

#ifndef __DYNAMIC_DATA_OBSERVED_E_IDL__
#define __DYNAMIC_DATA_OBSERVED_E_IDL__ 1
```

```
interface Dynamic_Data_Observed_E {

    typedef sequence<Dynamic_Data_Observer> Dynamic_Data_Observers;
    readonly attribute Dynamic_Data_Observers dynamic_data_observer_list;

    void attach_dynamic_data_observer(
        in Dynamic_Data_Observer obs);

    void detach_dynamic_data_observer(
        in Dynamic_Data_Observer obs)
        raises(Domain_Repository::Unknown_Observer);
};

#endif                                  /* __DYNAMIC_DATA_OBSERVED_E_IDL__ */

#ifndef __DYNAMIC_DATA_OBSERVER_IDL__
#define __DYNAMIC_DATA_OBSERVER_IDL__ 1

interface Dynamic_Data_Observer {
    oneway void receive_sighting(in Sighting_Description sd);

    oneway void abort();
};

#endif                                  /* __DYNAMIC_DATA_OBSERVER_IDL__ */
```

The similar scheme is used to inform applications about changes of a view definition. These changes comprise: adding a new location and locatable to the view and removing a location or a locatable from the view. (A view definition may be changed via a management interface described in the next point.)

```
#ifndef __VIEW_DATA_OBSERVED_IDL__
#define __VIEW_DATA_OBSERVED_IDL__ 1

interface View_Data_Observed {

    struct Observer_Entry {
        View_Data_Observer obs;                              // Observer reference
        Domain_Repository::Repository_Item_Ids interest;     // observer interest
    };

    typedef sequence<View_Data_Observer> View_Data_Observers;
    readonly attribute View_Data_Observers view_data_observer_list;

    void attach_view_data_observer(
        in View_Data_Observer obs);

    void detach_view_data_observer(
        in View_Data_Observer obs)
        raises(Domain_Repository::Unknown_Observer);
};
```

```
#endif                                        /* __VIEW_DATA_OBSERVED_IDL__ */

#ifndef __VIEW_DATA_OBSERVER_IDL__
#define __VIEW_DATA_OBSERVER_IDL__ 1

interface View_Data_Observer {

    enum View_Data_Type {
        LOCATION_VD,
        LOCATABLE_VD
    };

    union View_Data_Object switch (View_Data_Type) {
        case LOCATION_VD : Location location_obj;
        case LOCATABLE_VD : Locatable locatable_obj;
    };

    oneway void notify(
        in Domain_Repository::Repository_Item_Id id,
        in View_Data_Object sd,
        in Domain_Repository::Change_Type change);
    oneway void abort();
};

#endif                                        /* __VIEW_DATA_OBSERVER_IDL__ */
```

The location, locatable and view, which are the basic notions for ABng core clients, provide interfaces defined below.

Location is an object that represents the certain real location (described by the *Location_Description* object in the database layer) in a given view. The *get_sightings* method enable to obtain a list of sightings of locatables which are currently localized in the location. An application may attach an observer of dynamic data (via the *Dynamic_Data_Observed* base interface) if it wants to obtain sightings from that location asynchronously. Changes of static attributes of the location may be also reported asynchronously. The *Location_Commander* base interface allows to send commands to the location or to locatables in that location.

```
#ifndef __LOCATION_IDL__
#define __LOCATION_IDL__ 1

interface Location: Dynamic_Data_Observed_E,
    View_Data_Observed_E,
    Location_Commander {

    struct Description {
        Domain_Repository::Repository_Item_Id item_id;
        Domain_Repository::Location_Description loc_descr;
    };

    typedef sequence<Description> Descriptions;
```

```
    struct Value_Description {
        Domain_Repository::Repository_Item_Id item_id;
        Domain_Repository::Location_Description::Value_Description
            loc_value_descr;
    };

    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
        boolean is_any;
        Domain_Repository::Repository_Item_Id_Pattern item_id_pattern;
        Domain_Repository::Location_Description::Pattern loc_descr_pattern;
    };

    readonly attribute Domain_Repository::Repository_Item_Id item_id;
//    readonly attribute Description descr;
    Description descr()
        raises (Domain_Repository::Destination_Unreachable);
//    readonly attribute Value_Description value_descr;
    Value_Description value_descr()
        raises (Domain_Repository::Destination_Unreachable);
//    readonly attribute Domain_Repository::Location_Description loc_descr;
    Domain_Repository::Location_Description loc_descr()
        raises (Domain_Repository::Destination_Unreachable);

    Sighting_Descriptions get_sightings(in Time since);
};

#endif                                              /* __LOCATION_IDL__ */
```

Locatable is an object that represents the certain real locatable object (a user or a piece of equipment) in a given view. The *get_sighting* method enable to obtain information about current location of the locatable object (last sighting data). An application may also attach an observer of dynamic data (via the *Dynamic_Data_Observed* base interface) if it wants to obtain sightings from that locatable asynchronously. Changes of static attributes of the locatable may be also reported asynchronously. The *Locatable_Commander* base interface allows to send commands to the locatable in specified locations.

```
#ifndef __LOCATABLE_IDL__
#define __LOCATABLE_IDL__ 1

interface Locatable : Dynamic_Data_Observed_E,
    View_Data_Observed_E,
    Locatable_Commander {

    struct Description {
        Domain_Repository::Repository_Item_Id item_id;
        Location home_location;
        Locatable_Type_Specific_Object type_specific_object;
    };
```

```
    typedef sequence<Description> Descriptions;

    struct Value_Description {
        Domain_Repository::Repository_Item_Id item_id;
        Location::Value_Description home_location;
        Locatable_Type_Specific_Object_Value
            type_specific_object_value;
    };

    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
        boolean is_any;
        Domain_Repository::Repository_Item_Id_Pattern item_id_pattern;
        Location::Pattern home_location_pattern;
        Locatable_Type_Specific_Object_Pattern
            type_specific_object_pattern;
    };

    readonly attribute Domain_Repository::Repository_Item_Id item_id;
//  readonly attribute Description descr;
    Description descr()
        raises (Domain_Repository::Destination_Unreachable);
//  readonly attribute Value_Description value_descr;
    Value_Description value_descr()
        raises (Domain_Repository::Destination_Unreachable);
//  readonly attribute Location home_location;
    Location home_location()
        raises (Domain_Repository::Destination_Unreachable);
//  readonly attribute Locatable_Type_Specific_Object
//      type_specific_object;
    Locatable_Type_Specific_Object type_specific_object()
        raises (Domain_Repository::Destination_Unreachable);

    Sighting_Description get_sighting();
};

#endif                                        /* __LOCATABLE_IDL__ */
```

The *View* interface provides the information about the view definition to applications. An application may obtain lists of location and locatables that constitute the view as well as the type of locations belonging to the view. The list of sightings emitted by locatables belonging to the view may be obtained using the *get_sightings* operation. An application may register call-back interfaces within a view to obtain sightings and changes of static data related to all locations and locatables in the view or changes of view definition. (Alternatively, an application may register itself in individual locations or locatables as it was described above). An application may send a command to any set of locations of locatables in the view using the *Commander* base interface.

```
#ifndef __VIEW_IDL__
#define __VIEW_IDL__ 1

interface View : Dynamic_Data_Observed_R,
                 View_Data_Observed_R,
                 Commander {

    readonly attribute string view_name;                    // Unique view name
    // Location type for which a view has been created.
    readonly attribute Domain_Repository::Location_Type::Location_Type_Name type;
    readonly attribute Locations location_list;
    readonly attribute Locatables locatable_list;
    readonly attribute boolean wants_duplicates;

    Locations find_location(
        in Location::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented,
               Domain_Repository::Destination_Unreachable);
    Location::Value_Descriptions find_location_values(
        in Location::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented,
               Domain_Repository::Destination_Unreachable);

    Locatables find_locatable(
        in Locatable::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented,
               Domain_Repository::Destination_Unreachable);
    Locatable::Value_Descriptions find_locatable_values(
        in Locatable::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented,
               Domain_Repository::Destination_Unreachable);

    Sighting_Descriptions get_sightings(in Time since);

};

#endif                                                  /* __VIEW_IDL__ */
```

References of particular view objects are provided to applications through the *View_Rep* interface presented below.

```
#ifndef __VIEW_REP_IDL__
#define __VIEW_REP_IDL__ 1

interface View_Rep {

    typedef sequence<View> Views;
```

```
    Views find(in string name)
        raises(Domain_Repository::Search_Not_Implemented);

    Views get_all();
};

#endif                                            /* __VIEW_REP_IDL__ */
```

The *ABng_Domain* interface holds basic information about the domain (domain description and data used for construction of proper phone numbers and IP addresses for the domain). They are obtained from *Domain_Repository::Domain* during the system initialization.

```
#ifndef __ABNG_DOMAIN_IDL__
#define __ABNG_DOMAIN_IDL__ 1

interface ABng_Domain {

    readonly attribute Domain_Repository::Location_Description
        domain_description;
    readonly attribute Domain_Repository::Local_Dependence
        domain_local_dependence;
};

#endif                                            /* __ABNG_DOMAIN_IDL__ */
```

5.2 ABng Poller interfaces

The *ABng_Poller* module is defined as follows.

```
#ifndef __ABNG_POLLER_IDL__
#define __ABNG_POLLER_IDL__ 1

module ABng_Poller {
#include "sabpoller.idl"
#include "abpoller.idl"
#include "sabobserver.idl"
#include "abobserver.idl"
};

#endif                                            /* __ABNG_POLLER_IDL__ */
```

The first include file defines the basic poller interface which allows to attach and detach a poller observer and send a single badge or a sensor command. A command buffer has to be formatted according to the protocol described in [1].

```
#ifndef __SABPOLLER_IDL__
#define __SABPOLLER_IDL__ 1

interface Simple_AB_Observer;
```

```
interface Simple_AB_Poller {

    typedef sequence<octet> Octets;
    typedef Octets Message;

    attribute boolean click_counting_flag;
    attribute string name;

    boolean attach_observer(in Simple_AB_Observer obs);
    boolean dettach_observer(in Simple_AB_Observer obs);
    boolean attach_type_observer(in Simple_AB_Observer obs, in short type);
    boolean dettach_type_observer(in Simple_AB_Observer obs, in short type);
    boolean dettach_observer_all(in Simple_AB_Observer obs);

    void issue_command(in Message cmd);

};

#endif                                                  /* __SABPOLLER_IDL__ */
```

The next include file contain a more abstract poller interface. It provides separate methods for execution of particular commands.

```
#ifndef __ABPOLLER_IDL__
#define __ABPOLLER_IDL__ 1

#include "sabpoller.idl"

interface AB_Poller : Simple_AB_Poller {

    typedef unsigned short Sighting_Code;

    const Sighting_Code PERIODIC_BROADCAST = 0x80;
    const Sighting_Code MIDDLE_BUTTON = 0x81;
    const Sighting_Code SIDE_BUTTON = 0x82;
    const Sighting_Code BADGE_IN_FIELD = 0x84;
    const Sighting_Code MIDDLE_BUTTON_IN_FIELD = 0x85;
    const Sighting_Code SIDE_BUTTON_IN_FIELD = 0x86;

    typedef octet Byte;
    typedef octet Network_Id;
    typedef octet Station_Id;
    typedef octet Seq_Number;
    typedef sequence<octet> Octets;
    typedef Octets Message;
    typedef Octets Badge_Id;
    typedef Octets Address;
    typedef Octets Tones;
    typedef Octets Field_Id;
    struct Time_Stamp {
        unsigned long sec;
        unsigned long usec;
```

```
};

void challenge(in Network_Id network,
    in Station_Id station,
    in Badge_Id badge,
    in Octets random,
    in Seq_Number seq);

void sound(in Network_Id network,
    in Station_Id station,
    in Badge_Id badge,
    in Tones sound,
    in Seq_Number seq);

void set_state(in Network_Id network,
    in Station_Id station,
    in Badge_Id badge,
    in Byte status,
    in Tones sound,
    in Seq_Number seq);

void get_state(in Network_Id network,
    in Station_Id station,
    in Badge_Id badge,
    in Seq_Number seq);

void set_home(in Network_Id network,
    in Station_Id station,
    in Badge_Id badge,
    in Address address,
    in Seq_Number seq);

void get_home(in Network_Id network,
    in Station_Id station,
    in Badge_Id badge,
    in Seq_Number seq);

void reset_buffer(in Network_Id network,
    in Station_Id station);

void reset_fifos(in Network_Id network,
    in Station_Id station);

void get_station_state(in Network_Id network,
    in Station_Id station);

void set_output(in Network_Id network,
    in Station_Id station,
    in Byte output_value);

void set_persistence(in Network_Id network,
    in Station_Id station,
```

```
            in boolean per_mode_value);

        void broadcast(in Network_Id network,
            in Station_Id station,
            in Octets data);

};

#endif                                              /* __ABPOLLER_IDL__ */
```

The *Simple_AB_Observer* interface defined below is provided by poller observers which want to receive raw sighting messages and command replies.

```
#ifndef __SABOBSERVER_IDL__
#define __SABOBSERVER_IDL__ 1

#include "abpoller.idl"

interface Simple_AB_Observer {

    oneway void receive(
        in string network_group_name,
        in AB_Poller::Message msg);

};

#endif                                              /* __SABOBSERVER_IDL__ */
```

The more abstract *AB_Observer* interface defined below provides decoded sighting messages and command replies to an observing application.

```
#ifndef __ABOBSERVER_IDL__
#define __ABOBSERVER_IDL__ 1

#include "sabobserver.idl"

interface AB_Observer : Simple_AB_Observer {

    void sighting(
        in string network_group_name,
        in AB_Poller::Network_Id network,
        in AB_Poller::Station_Id station,
        in AB_Poller::Badge_Id badge,
        in AB_Poller::Sighting_Code code,
        in AB_Poller::Field_Id field,
        in AB_Poller::Byte click_num,
        in AB_Poller::Seq_Number seq,
        in AB_Poller::Time_Stamp ts);

    void challenge_reply(
        in string network_group_name,
        in AB_Poller::Network_Id network,
```

```
        in AB_Poller::Station_Id station,
        in AB_Poller::Badge_Id badge,
        in AB_Poller::Octets random,
        in AB_Poller::Seq_Number seq,
        in AB_Poller::Time_Stamp ts);

void sound_reply(
        in string network_group_name,
        in AB_Poller::Network_Id network,
        in AB_Poller::Station_Id station,
        in AB_Poller::Badge_Id badge,
        in AB_Poller::Tones sound,
        in AB_Poller::Seq_Number seq,
        in AB_Poller::Time_Stamp ts);

void set_state_reply(
        in string network_group_name,
        in AB_Poller::Network_Id network,
        in AB_Poller::Station_Id station,
        in AB_Poller::Badge_Id badge,
        in AB_Poller::Byte status,
        in AB_Poller::Tones sound,
        in AB_Poller::Seq_Number seq,
        in AB_Poller::Time_Stamp ts);

void get_state_reply(
        in string network_group_name,
        in AB_Poller::Network_Id network,
        in AB_Poller::Station_Id station,
        in AB_Poller::Badge_Id badge,
        in AB_Poller::Byte major_version,
        in AB_Poller::Byte minor_version,
        in AB_Poller::Byte status,
        in AB_Poller::Seq_Number seq,
        in AB_Poller::Time_Stamp ts);

void set_home_reply(
        in string network_group_name,
        in AB_Poller::Network_Id network,
        in AB_Poller::Station_Id station,
        in AB_Poller::Badge_Id badge,
        in AB_Poller::Address address,
        in AB_Poller::Seq_Number seq,
        in AB_Poller::Time_Stamp ts);

void get_home_reply(
        in string network_group_name,
        in AB_Poller::Network_Id network,
        in AB_Poller::Station_Id station,
        in AB_Poller::Badge_Id badge,
        in AB_Poller::Address address,
        in AB_Poller::Seq_Number seq,
```

```
            in AB_Poller::Time_Stamp ts);

        void get_station_state_reply(
            in string network_group_name,
            in AB_Poller::Network_Id network,
            in AB_Poller::Station_Id station,
            in AB_Poller::Byte major_version,
            in AB_Poller::Byte minor_version,
            in AB_Poller::Byte status1,
            in AB_Poller::Byte status2,
            in AB_Poller::Time_Stamp ts);

        void null_message(
            in string network_group_name,
            in AB_Poller::Network_Id network,
            in AB_Poller::Station_Id station,
            in AB_Poller::Time_Stamp ts);

    };

    #endif                                        /* __ABOBSERVER_IDL__ */

    //
```

5.3 ABng System Interfaces

The *ABng_Core_System* module is defined as follows.

```
#include <orb.idl>

#include "abng_core.idl"
#include "abng_poller.idl"


module ABng_Core_System {

interface System_Static_Data_Observer ;
interface System_Static_Data_Observed_E ;
interface System_Static_Data_Observed_R ;
interface Scheduler_Reqeust_Queue ;
interface Scheduler_Caller ;
interface Scheduler ;
interface Badge_Commander ;
interface Badge ;
interface Badge_Holder ;
interface Sensor ;
interface Sensor_Commander ;
interface ABng_Location_Description ;

typedef Domain_Repository :: Octets Octets ;
typedef Domain_Repository :: Octets_Pattern Octets_Pattern ;
```

```
typedef Domain_Repository :: Time Time ;

typedef Octets Badge_Id ;
typedef Octets_Pattern Badge_Id_Pattern ;

typedef unsigned short Sensor_Number ;
typedef unsigned short Network_Number ;
typedef string Network_Group_Name ;
typedef Domain_Repository :: Unsigned_Short_Pattern Sensor_Number_Pattern ;
typedef Domain_Repository :: Unsigned_Short_Pattern Network_Number_Pattern ;
typedef Domain_Repository :: String_Pattern Network_Group_Name_Pattern ;

struct Sensor_Id {
Sensor_Number sensor_num ;
Network_Number network_num ;
Network_Group_Name group_name ;
} ;

struct Sensor_Id_Pattern {
boolean is_any ;
Sensor_Number_Pattern sensor_number_pattern ;
Network_Number_Pattern network_number_pattern ;
Network_Group_Name_Pattern network_group_name_pattern ;
} ;

typedef sequence < Sensor > Sensors ;
typedef sequence < Badge > Badges ;
typedef sequence < Badge_Holder > Badge_Holders ;
typedef sequence < ABng_Location_Description > ABng_Location_Descriptions ;
typedef sequence < Sensor_Id > Sensor_Ids ;
typedef sequence < Badge_Id > Badge_Ids ;

exception Unknown_Sensor_Id {
Sensor_Id sid ;
} ;

exception Invalid_Sensor_Id {
Sensor_Id sid ;
} ;

exception Unknown_Badge_Id {
Badge_Id bid ;
} ;

exception Invalid_Badge_Id {
Badge_Id bid ;
} ;

interface System_Static_Data_Observed_E {

typedef sequence < System_Static_Data_Observer > System_Static_Data_Observers ;
readonly attribute System_Static_Data_Observers system_static_data_observer_list ;
```

```
void attach_system_static_data_observer (
in System_Static_Data_Observer obs ) ;

void detach_system_static_data_observer (
in System_Static_Data_Observer obs )
raises ( Domain_Repository :: Unknown_Observer ) ;
} ;

interface System_Static_Data_Observed_R {

struct Observer_Entry {
System_Static_Data_Observer obs ;
Domain_Repository :: Repository_Item_Ids interest ;
} ;

typedef sequence < Observer_Entry > Observer_Entries ;
readonly attribute Observer_Entries system_static_data_observer_list ;

readonly attribute System_Static_Data_Observed_E :: System_Static_Data_Observers
add_system_static_data_observer_list ;

void attach_system_static_data_observer (
in System_Static_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ) ;

void detach_system_static_data_observer (
in System_Static_Data_Observer obs ,
in Domain_Repository :: Repository_Item_Ids observed_id_list )
raises ( Domain_Repository :: Unknown_Repository_Item_Ids ,
Domain_Repository :: Unknown_Observer ,
Domain_Repository :: Unmatched_Observer_Item_Ids ) ;

void attach_add_system_static_data_observer (
in System_Static_Data_Observer obs ) ;

void detach_add_system_static_data_observer (
in System_Static_Data_Observer obs )
raises ( Domain_Repository :: Unknown_Observer ) ;
} ;

interface System_Static_Data_Observer {

enum System_Static_Data_Type {
SENSOR_SD ,
BADGE_SD ,
BADGE_HOLDER_SD ,
ABNG_LOCATION_DESCRIPTION_SD
} ;

union System_Static_Data_Object switch ( System_Static_Data_Type ) {
```

```
case SENSOR_SD : Sensor sensor_obj ;
case BADGE_SD : Badge badge_obj ;
case BADGE_HOLDER_SD : Badge_Holder badge_holder_obj ;
case ABNG_LOCATION_DESCRIPTION_SD : ABng_Location_Description
abng_loc_descr_obj ;
} ;

oneway void notify (
in Domain_Repository :: Repository_Item_Id id ,
in System_Static_Data_Object sd ,
in Domain_Repository :: Change_Type change ) ;
oneway void abort ( ) ;
} ;

interface Scheduler_Request_Queue {

struct Scheduler_Request {
string uid ;
long request_id ;
Sensor_Ids sensors ;
Badge_Ids badges ;
Octets arguments ;
string command_name ;
ABng_Core :: Command_Context command_context ;
Time request_time ;
} ;

typedef sequence < Scheduler_Request > Scheduler_Requests ;

Scheduler_Requests list_user_requests ( in string user_id )
raises ( Domain_Repository :: Destination_Unreachable ) ;
Scheduler_Requests list_all_requests ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
void remove_request ( in long request_id )
raises ( ABng_Core :: Unknown_Request_Id ,
Domain_Repository :: Destination_Unreachable ) ;
} ;

interface Scheduler_Caller {

oneway void challenge_reply (
in Sensor_Id sensor ,
in Badge_Id badge ,
in Octets random ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void sound_reply (
in Sensor_Id sensor ,
in Badge_Id badge ,
in Octets sound ,
```

```
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void set_state_reply (
in Sensor_Id sensor ,
in Badge_Id badge ,
in octet status ,
in Octets sound ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void get_state_reply (
in Sensor_Id sensor ,
in Badge_Id badge ,
in octet major_version ,
in octet minor_version ,
in octet status ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void set_home_reply (
in Sensor_Id sensor ,
in Badge_Id badge ,
in Octets address ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void get_home_reply (
in Sensor_Id sensor ,
in Badge_Id badge ,
in Octets address ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void get_station_state_reply (
in Sensor_Id sensor ,
in octet major_version ,
in octet minor_version ,
in octet status1 ,
in octet status2 ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void sensor_command_reply (
in Sensor_Id sensor ,
in Octets reply_body ,
```

```
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void badge_command_reply (
in Sensor_Id sensor ,
in Badge_Id badge ,
in Octets reply_body ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void null_message (
in Sensor_Id sensor ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void request_processed (
in Sensor_Id sensor ,
in Badge_Id badge ,
in Time ts ,
in long request_id ,
in long repeat_num ) ;

oneway void error_occured (
in Sensor_Id sensor ,
in Badge_Id badge ,
in ABng_Core :: Command_Error_Code error_code ,
in long request_id ,
in long repeat_num ,
in long retry_num ) ;

oneway void request_completed (
in long request_id ) ;
} ;

interface Scheduler : Scheduler_Request_Queue {


long issue_sensor_command (
in Sensor_Ids sensors ,
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;
```

```
long issue_badge_command (
in Sensor_Ids sensors ,
in Badge_Ids badges ,
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long challenge (
in Sensor_Ids sensors ,
in Badge_Ids badges ,
in Octets random ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long sound (
in Sensor_Ids sensors ,
in Badge_Ids badges ,
in Octets sound ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long set_state (
in Sensor_Ids sensors ,
in Badge_Ids badges ,
in octet status ,
in Octets sound ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
```

```
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long get_state (
in Sensor_Ids sensors ,
in Badge_Ids badges ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long set_home (
in Sensor_Ids sensors ,
in Badge_Ids badges ,
in Octets address ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long get_home (
in Sensor_Ids sensors ,
in Badge_Ids badges ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long reset_buffer (
in Sensor_Ids sensors ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
```

```
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long reset_fifos (
in Sensor_Ids sensors ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long get_station_state (
in Sensor_Ids sensors ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long set_output (
in Sensor_Ids sensors ,
in octet output_value ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;

long set_persistence (
in Sensor_Ids sensors ,
in boolean mode ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( Unknown_Sensor_Id ,
Invalid_Sensor_Id ,
Unknown_Badge_Id ,
Invalid_Badge_Id ,
ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ) ;
} ;

interface Sensor_Commander : Scheduler_Request_Queue {
```

```
long issue_sensor_command (
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long issue_badge_command (
in Badges badges ,
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long challenge (
in Badges badges ,
in Octets random ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long sound (
in Badges badges ,
in Octets sound ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_state (
in Badges badges ,
in octet status ,
in Octets sound ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_state (
in Badges badges ,
in ABng_Core :: Command_Context ctx ,
```

```
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_home (
in Badges badges ,
in Octets address ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_home (
in Badges badges ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;


long reset_buffer (
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long reset_fifos (
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_station_state (
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_output (
in octet output_value ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;
```

```
long set_persistence (
in boolean mode ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

} ;

interface Sensor : System_Static_Data_Observed_E , Sensor_Commander {

struct Description {
Domain_Repository :: Repository_Item_Id item_id ;
Sensor_Id id ;
} ;

typedef sequence < Description > Descriptions ;

typedef Description Value_Description ;
typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
Domain_Repository :: Repository_Item_Id_Pattern item_id_pattern ;
Sensor_Id_Pattern id_pattern ;
} ;



struct Sensors_Pattern {
boolean is_any ;
Domain_Repository :: Sequence_Pattern_Constraints constraints ;
sequence < Pattern > sensors_pattern ;
} ;

readonly attribute Description descr ;
readonly attribute Value_Description value_descr ;
readonly attribute Domain_Repository :: Repository_Item_Id item_id ;
readonly attribute Sensor_Id id ;
} ;

interface Badge_Commander : Scheduler_Request_Queue {

long issue_command (
in Sensors sensors ,
in Octets cmd ,
in boolean has_reply ,
in octet reply_code ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
```

```
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long challenge (
in Sensors sensors ,
in Octets random ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long sound (
in Sensors sensors ,
in Octets sound ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_state (
in Sensors sensors ,
in octet status ,
in Octets sound ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_state (
in Sensors sensors ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long set_home (
in Sensors sensors ,
in Octets address ,
in ABng_Core :: Command_Context ctx ,
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;

long get_home (
in Sensors sensors ,
in ABng_Core :: Command_Context ctx ,
```

```
in Scheduler_Caller reply_to )
raises ( ABng_Core :: Invalid_Request_Time ,
ABng_Core :: Invalid_Argument ,
Domain_Repository :: Destination_Unreachable ) ;
} ;

interface Badge : System_Static_Data_Observed_E , Badge_Commander {

struct Description {
Domain_Repository :: Repository_Item_Id item_id ;
Badge_Id id ;
Badge_Holder holder ;
} ;

typedef sequence < Description > Descriptions ;

typedef Description Value_Description ;
typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
Domain_Repository :: Repository_Item_Id_Pattern item_id_pattern ;
Badge_Id_Pattern id_pattern ;
} ;

Description descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
Value_Description value_descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
readonly attribute Domain_Repository :: Repository_Item_Id item_id ;
readonly attribute Badge_Id id ;
Badge_Holder holder ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
} ;

interface Badge_Holder : System_Static_Data_Observed_E {

struct Description {
Domain_Repository :: Repository_Item_Id item_id ;
Badge my_badge ;
Badge current_badge ;
Sensor home_sensor ;
ABng_Core :: Locatable_Type_Specific_Object type_specific_object ;
} ;

typedef sequence < Description > Descriptions ;

struct Value_Description {
Domain_Repository :: Repository_Item_Id item_id ;
Badge :: Value_Description my_badge ;
Badge :: Value_Description current_badge ;
Sensor :: Value_Description home_sensor ;
```

```
ABng_Core :: Locatable_Type_Specific_Object_Value type_specific_object ;
} ;

typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
Domain_Repository :: Repository_Item_Id_Pattern item_id_pattern ;
Badge :: Pattern my_badge_pattern ;
Badge :: Pattern current_badge_pattern ;
Sensor :: Pattern home_sensor_pattern ;
ABng_Core :: Locatable_Type_Specific_Object_Pattern
type_specific_object_pattern ;
} ;

Description descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
Value_Description value_descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
readonly attribute Domain_Repository :: Repository_Item_Id item_id ;
Badge my_badge ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
Badge current_badge ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
Sensor home_sensor ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
ABng_Core :: Locatable_Type_Specific_Object type_specific_object ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;

} ;

interface ABng_Location_Description : System_Static_Data_Observed_E {

struct Description {
Domain_Repository :: Repository_Item_Id item_id ;
Domain_Repository :: Location_Description loc_descr ;
Sensors sensor_list ;
} ;

typedef sequence < Description > Descriptions ;

struct Value_Description {
Domain_Repository :: Repository_Item_Id item_id ;
Domain_Repository :: Location_Description :: Value_Description
loc_descr ;
Sensor :: Value_Descriptions sensor_list ;
} ;

typedef sequence < Value_Description > Value_Descriptions ;

struct Pattern {
boolean is_any ;
```

```
Domain_Repository :: Repository_Item_Id_Pattern item_id_pattern ;
Domain_Repository :: Location_Description :: Pattern
loc_descr_pattern ;
Sensor :: Sensors_Pattern sensor_list_pattern ;
} ;

Description descr ( ) raises ( Domain_Repository :: Destination_Unreachable ) ;
Value_Description value_descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;

readonly attribute Domain_Repository :: Repository_Item_Id item_id ;

Domain_Repository :: Location_Description loc_descr ( )
raises ( Domain_Repository :: Destination_Unreachable ) ;
Sensors sensor_list ( ) raises ( Domain_Repository :: Destination_Unreachable ) ;

void add ( in Sensor new_sensor )
raises ( Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Unknown_Object_Ref ) ;
void remove ( in Sensor sensor_to_remove )
raises ( Domain_Repository :: Unknown_Object_Ref ) ;
} ;

interface ABng_Domain_System {

readonly attribute ABng_Core :: ABng_Domain domain ;
readonly attribute Scheduler my_scheduler ;
} ;

} ;
```

The IDL files included in the above definition are presented and described below. The *abng_core_system_types.idl* file contains common type definitions including types of badge and sensor identifiers.

```
#ifndef __ABNG_CORE_SYSTEM_TYPES_IDL__
#define __ABNG_CORE_SYSTEM_TYPES_IDL__ 1

interface System_Static_Data_Observer;
interface System_Static_Data_Observed_E;
interface System_Static_Data_Observed_R;
interface Scheduler_Reqeust_Queue;
interface Scheduler_Caller;
interface Scheduler;
interface Badge_Commander;
interface Badge;
interface Badge_Holder;
interface Sensor;
interface Sensor_Commander;
interface ABng_Location_Description;

typedef Domain_Repository::Octets Octets;
```

```
typedef Domain_Repository::Octets_Pattern Octets_Pattern;
typedef Domain_Repository::Time Time;

typedef Octets Badge_Id;
typedef Octets_Pattern Badge_Id_Pattern;

typedef unsigned short Sensor_Number;
typedef unsigned short Network_Number;
typedef string Network_Group_Name;
typedef Domain_Repository::Unsigned_Short_Pattern Sensor_Number_Pattern;
typedef Domain_Repository::Unsigned_Short_Pattern Network_Number_Pattern;
typedef Domain_Repository::String_Pattern Network_Group_Name_Pattern;

struct Sensor_Id {
    Sensor_Number sensor_num;
    Network_Number network_num;
    Network_Group_Name group_name;
};

struct Sensor_Id_Pattern {
    boolean is_any;
    Sensor_Number_Pattern sensor_number_pattern;
    Network_Number_Pattern network_number_pattern;
    Network_Group_Name_Pattern network_group_name_pattern;
};

typedef sequence<Sensor> Sensors;
typedef sequence <Badge> Badges;
typedef sequence<Badge_Holder> Badge_Holders;
typedef sequence<ABng_Location_Description> ABng_Location_Descriptions;
typedef sequence<Sensor_Id> Sensor_Ids;
typedef sequence<Badge_Id> Badge_Ids;

exception Unknown_Sensor_Id {
    Sensor_Id sid;
};

exception Invalid_Sensor_Id {
    Sensor_Id sid;
};

exception Unknown_Badge_Id {
    Badge_Id bid;
};

exception Invalid_Badge_Id {
    Badge_Id bid;
};

#endif                                  /* __ABNG_CORE_SYSTEM_TYPES_IDL__ */
```

The next four include files define interfaces used for asynchronous notification about changes of system static data.

```
#ifndef __SYSTEM_STATIC_DATA_OBSERVED_R_IDL__
#define __SYSTEM_STATIC_DATA_OBSERVED_R_IDL__ 1

interface System_Static_Data_Observed_R {

    struct Observer_Entry {
        System_Static_Data_Observer obs;                          // Observer reference
        Domain_Repository::Repository_Item_Ids interest;          // observer interest
    };

    typedef sequence<Observer_Entry> Observer_Entries;
    readonly attribute Observer_Entries system_static_data_observer_list;

    readonly attribute System_Static_Data_Observed_E::System_Static_Data_Observers
                add_system_static_data_observer_list;

    void attach_system_static_data_observer(
        in System_Static_Data_Observer obs,
        in Domain_Repository::Repository_Item_Ids observed_id_list)
        raises(Domain_Repository::Unknown_Repository_Item_Ids);

    void detach_system_static_data_observer(
        in System_Static_Data_Observer obs,
        in Domain_Repository::Repository_Item_Ids observed_id_list)
        raises(Domain_Repository::Unknown_Repository_Item_Ids,
               Domain_Repository::Unknown_Observer,
               Domain_Repository::Unmatched_Observer_Item_Ids);

    void attach_add_system_static_data_observer(
        in System_Static_Data_Observer obs);

    void detach_add_system_static_data_observer(
        in System_Static_Data_Observer obs)
        raises(Domain_Repository::Unknown_Observer);
};

#endif                     /* __SYSTEM_STATIC_DATA_OBSERVED_R_IDL__ */

#ifndef __SYSTEM_STATIC_DATA_OBSERVED_E_IDL__
#define __SYSTEM_STATIC_DATA_OBSERVED_E_IDL__ 1

interface System_Static_Data_Observed_E {

    typedef sequence<System_Static_Data_Observer> System_Static_Data_Observers;
    readonly attribute System_Static_Data_Observers system_static_data_observer_list;

    void attach_system_static_data_observer(
        in System_Static_Data_Observer obs);

    void detach_system_static_data_observer(
        in System_Static_Data_Observer obs)
        raises(Domain_Repository::Unknown_Observer);
};
```

```
#endif                          /* __SYSTEM_STATIC_DATA_OBSERVED_E_IDL__ */

#ifndef __SYSTEM_STATIC_DATA_OBSERVER_IDL__
#define __SYSTEM_STATIC_DATA_OBSERVER_IDL__ 1

interface System_Static_Data_Observer {

    enum System_Static_Data_Type {
        SENSOR_SD,
        BADGE_SD,
        BADGE_HOLDER_SD,
        ABNG_LOCATION_DESCRIPTION_SD
    };

    union System_Static_Data_Object switch (System_Static_Data_Type) {
        case SENSOR_SD : Sensor sensor_obj;
        case BADGE_SD : Badge badge_obj;
        case BADGE_HOLDER_SD : Badge_Holder badge_holder_obj;
        case ABNG_LOCATION_DESCRIPTION_SD : ABng_Location_Description
            abng_loc_descr_obj;
    };

    oneway void notify(
        in Domain_Repository::Repository_Item_Id id,
        in System_Static_Data_Object sd,
        in Domain_Repository::Change_Type change);
    oneway void abort();
};

#endif                          /* __SYSTEM_STATIC_DATA_OBSERVER_IDL__ */
```

The next includes contain definitions of interfaces used to send badge and sensor commands. They are similar to high-level command interfaces described above.

```
#ifndef __SCHEDULER_REQUEST_QUEUE_IDL__
#define __SCHEDULER_REQUEST_QUEUE_IDL__ 1

interface Scheduler_Request_Queue {

    struct Scheduler_Request {
        string uid;
        long request_id;
        Sensor_Ids sensors;
        Badge_Ids badges;
        Octets arguments;
        string command_name;
        ABng_Core::Command_Context command_context;
        Time request_time;
    };

    typedef sequence<Scheduler_Request> Scheduler_Requests;
```

```
    Scheduler_Requests list_user_requests(in string user_id)
        raises (Domain_Repository::Destination_Unreachable);
    Scheduler_Requests list_all_requests()
        raises (Domain_Repository::Destination_Unreachable);
    void remove_request(in long request_id)
        raises (ABng_Core::Unknown_Request_Id,
                Domain_Repository::Destination_Unreachable);
};

#endif                                  /* __SCHEDULER_REQUEST_QUEUE_IDL__ */

#ifndef __SCHEDULER_CALLER_IDL__
#define __SCHEDULER_CALLER_IDL__ 1

// The interface used for sending command replies. Should be provided
// by a command caller.

interface Scheduler_Caller {

    oneway void challenge_reply(
        in Sensor_Id sensor,
        in Badge_Id badge,
        in Octets random,
        in Time ts,
        in long request_id,
        in long repeat_num);

    oneway void sound_reply(
        in Sensor_Id sensor,
        in Badge_Id badge,
        in Octets sound,
        in Time ts,
        in long request_id,
        in long repeat_num);

    oneway void set_state_reply(
        in Sensor_Id sensor,
        in Badge_Id badge,
        in octet status,
        in Octets sound,
        in Time ts,
        in long request_id,
        in long repeat_num);

    oneway void get_state_reply(
        in Sensor_Id sensor,
        in Badge_Id badge,
        in octet major_version,
        in octet minor_version,
        in octet status,
        in Time ts,
        in long request_id,
```

```
    in long repeat_num);

oneway void set_home_reply(
    in Sensor_Id sensor,
    in Badge_Id badge,
    in Octets address,
    in Time ts,
    in long request_id,
    in long repeat_num);

oneway void get_home_reply(
    in Sensor_Id sensor,
    in Badge_Id badge,
    in Octets address,
    in Time ts,
    in long request_id,
    in long repeat_num);

oneway void get_station_state_reply(
    in Sensor_Id sensor,
    in octet major_version,
    in octet minor_version,
    in octet status1,
    in octet status2,
    in Time ts,
    in long request_id,
    in long repeat_num);

oneway void sensor_command_reply(
    in Sensor_Id sensor,
    in Octets reply_body,
    in Time ts,
    in long request_id,
    in long repeat_num);

oneway void badge_command_reply(
    in Sensor_Id sensor,
    in Badge_Id badge,
    in Octets reply_body,
    in Time ts,
    in long request_id,
    in long repeat_num);

oneway void null_message(
    in Sensor_Id sensor,
    in Time ts,
    in long request_id,
    in long repeat_num);

oneway void request_processed(
    in Sensor_Id sensor,
    in Badge_Id badge,
```

```
        in Time ts,
        in long request_id,
        in long repeat_num);

    oneway void error_occured(
        in Sensor_Id sensor,
        in Badge_Id badge,
        in ABng_Core::Command_Error_Code error_code,
        in long request_id,
        in long repeat_num,
        in long retry_num);

    oneway void request_completed(
        in long request_id);
};

#endif                                          /* __SCHEDULER_CALLER_IDL__ */

#ifndef __SCHEDULER_IDL__
#define __SCHEDULER_IDL__ 1

#include "abs_types.idl"
#include "abobserver.idl"

// AB_Scheduler
interface Scheduler : AB_Observer {

    struct Request {

        string uid;
        long request_id;
        string command_name;
        Command_Context command_context;
        Time request_time;
    };

    typedef sequence<Request> Requests;

    Requests list_user_request(in string user_id);
    Requests list_all_requests();
    boolean remove_request(in long request_id);

    // Send to sensors
    long issue_sensor_command(
        in Sensors sensors,
        in Octets cmd,
        in Command_Context ctx);

    // Send through the specified sensors to badges.
    // If there are no sensors, send through any sensor.
    // If there are no badges, send to all badges.
    long issue_badge_command(in Sensors sensors,
        in Badges badges,
```

```
        in Octets cmd,
        in Command_Context ctx);

    long challenge(
        in Badges badges,
        in Sensors sensors,
        in Octets random,
        in Command_Context ctx);

    long sound(
        in Badges badges,
        in Sensors sensors,
        in Octets sound,
        in Command_Context ctx);

    long set_state(
        in Badges badges,
        in Sensors sensors,
        in octet status,
        in Octets sound,
        in Command_Context ctx);

    long get_state(
        in Badges badges,
        in Sensors sensors,
        in Command_Context ctx);

    long set_home(
        in Badges badges,
        in Sensors sensors,
        in Octets address,
        in Command_Context ctx);

    long get_home(
        in Badges badges,
        in Sensors sensors,
        in Command_Context ctx);

    long reset_buffer(
        in Sensors sensors,
        in Command_Context ctx);

    long reset_fifos(
        in Sensors sensors,
        in Command_Context ctx);

    long get_station_state(
        in Sensors sensors,
        in Command_Context ctx);

    long set_output(
        in Sensors sensors,
```

```
        in octet output_value,
        in Command_Context ctx);

    long set_persistence(
        in Sensors sensors,
        in boolean mode,
        in Command_Context ctx);

};

#endif                                          /* __SCHEDULER_IDL__ */

#ifndef __SENSOR_COMMANDER_IDL__
#define __SENSOR_COMMANDER_IDL__ 1

interface Sensor_Commander : Scheduler_Request_Queue {

    long issue_sensor_command(
        in Octets cmd,
        in boolean has_reply,
        in octet reply_code,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long issue_badge_command(
        in Badges badges,
        in Octets cmd,
        in boolean has_reply,
        in octet reply_code,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long challenge(
        in Badges badges,
        in Octets random,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long sound(
        in Badges badges,
        in Octets sound,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
```

```
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

long set_state(
      in Badges badges,
      in octet status,
      in Octets sound,
      in ABng_Core::Command_Context ctx,
      in Scheduler_Caller reply_to)
      raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

long get_state(
      in Badges badges,
      in ABng_Core::Command_Context ctx,
      in Scheduler_Caller reply_to)
      raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

long set_home(
      in Badges badges,
      in Octets address,
      in ABng_Core::Command_Context ctx,
      in Scheduler_Caller reply_to)
      raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

long get_home(
      in Badges badges,
      in ABng_Core::Command_Context ctx,
      in Scheduler_Caller reply_to)
      raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

// Sensor commands
long reset_buffer(
      in ABng_Core::Command_Context ctx,
      in Scheduler_Caller reply_to)
      raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

long reset_fifos(
      in ABng_Core::Command_Context ctx,
      in Scheduler_Caller reply_to)
      raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);
```

```
    long get_station_state(
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long set_output(
        in octet output_value,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

    long set_persistence(
        in boolean mode,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
            ABng_Core::Invalid_Argument,
            Domain_Repository::Destination_Unreachable);

};

#endif                                          /* __SENSOR_COMMANDER_IDL__ */

#ifndef __BADGE_COMMANDER_IDL__
#define __BADGE_COMMANDER_IDL__ 1

interface Badge_Commander : Scheduler_Request_Queue {

    long issue_command(
        in Sensors sensors,
        in Octets cmd,
        in boolean has_reply,
        in octet reply_code,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
                ABng_Core::Invalid_Argument,
                Domain_Repository::Destination_Unreachable);

    long challenge(
        in Sensors sensors,
        in Octets random,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
                ABng_Core::Invalid_Argument,
                Domain_Repository::Destination_Unreachable);
```

```
    long sound(
        in Sensors sensors,
        in Octets sound,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
                ABng_Core::Invalid_Argument,
                Domain_Repository::Destination_Unreachable);

    long set_state(
        in Sensors sensors,
        in octet status,
        in Octets sound,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
                ABng_Core::Invalid_Argument,
                Domain_Repository::Destination_Unreachable);

    long get_state(
        in Sensors sensors,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
                ABng_Core::Invalid_Argument,
                Domain_Repository::Destination_Unreachable);

    long set_home(
        in Sensors sensors,
        in Octets address,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
                ABng_Core::Invalid_Argument,
                Domain_Repository::Destination_Unreachable);

    long get_home(
        in Sensors sensors,
        in ABng_Core::Command_Context ctx,
        in Scheduler_Caller reply_to)
        raises (ABng_Core::Invalid_Request_Time,
                ABng_Core::Invalid_Argument,
                Domain_Repository::Destination_Unreachable);
};

#endif                                    /* __BADGE_COMMANDER_IDL__ */
```

Next two files contain interfaces for two basic system components: a badge and a sensor.

```
#ifndef __SENSOR_IDL__
#define __SENSOR_IDL__ 1

interface Sensor: System_Static_Data_Observed_E, Sensor_Commander {
```

```
    struct Description {
        Domain_Repository::Repository_Item_Id item_id;
        Sensor_Id id;
    };

    typedef sequence<Description> Descriptions;

    typedef Description Value_Description;
    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
        boolean is_any;
        Domain_Repository::Repository_Item_Id_Pattern item_id_pattern;
        Sensor_Id_Pattern id_pattern;
    };

    // Yes, yes.... I know you don't want it here....
    // But try to compile my rep. without this definition :(
    // G.
    struct Sensors_Pattern {
        boolean is_any;
        Domain_Repository::Sequence_Pattern_Constraints constraints;
        sequence<Pattern> sensors_pattern;
    };

    readonly attribute Description descr;
    readonly attribute Value_Description value_descr;
    readonly attribute Domain_Repository::Repository_Item_Id item_id;
    readonly attribute Sensor_Id id;
};

#endif                                                  /* __SENSOR_IDL__ */


#ifndef __BADGE_IDL__
#define __BADGE_IDL__ 1

interface Badge: System_Static_Data_Observed_E, Badge_Commander {

    struct Description {
        Domain_Repository::Repository_Item_Id item_id;
        Badge_Id id;
        Badge_Holder holder;
    };

    typedef sequence<Description> Descriptions;

    typedef Description Value_Description;
    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
        boolean is_any;
```

```
        Domain_Repository::Repository_Item_Id_Pattern item_id_pattern;
        Badge_Id_Pattern id_pattern;
    };

    Description descr()
        raises (Domain_Repository::Destination_Unreachable);
    Value_Description value_descr()
        raises (Domain_Repository::Destination_Unreachable);
    readonly attribute Domain_Repository::Repository_Item_Id item_id;
    readonly attribute Badge_Id id;
    Badge_Holder holder()
        raises (Domain_Repository::Destination_Unreachable);
};

#endif                                              /* __BADGE_IDL__ */
```

The *Badge_Holder* interface defined below holds information about bindings of a given badge to a user or a piece of equipment.

```
#ifndef __BADGE_HOLDER_IDL__
#define __BADGE_HOLDER_IDL__ 1

interface Badge_Holder : System_Static_Data_Observed_E {

    struct Description {
        Domain_Repository::Repository_Item_Id item_id;
        Badge my_badge;
        Badge current_badge;
        Sensor home_sensor;
        ABng_Core::Locatable_Type_Specific_Object type_specific_object;
    };

    typedef sequence<Description> Descriptions;

    struct Value_Description {
        Domain_Repository::Repository_Item_Id item_id;
        Badge::Value_Description my_badge;
        Badge::Value_Description current_badge;
        Sensor::Value_Description home_sensor;
        ABng_Core::Locatable_Type_Specific_Object_Value type_specific_object;
    };

    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
        boolean is_any;
        Domain_Repository::Repository_Item_Id_Pattern item_id_pattern;
        Badge::Pattern my_badge_pattern;
        Badge::Pattern current_badge_pattern;
        Sensor::Pattern home_sensor_pattern;
        ABng_Core::Locatable_Type_Specific_Object_Pattern
            type_specific_object_pattern;
```

```
    };

    Description descr()
        raises (Domain_Repository::Destination_Unreachable);
    Value_Description value_descr()
        raises (Domain_Repository::Destination_Unreachable);
    readonly attribute Domain_Repository::Repository_Item_Id item_id;
    Badge my_badge()
        raises (Domain_Repository::Destination_Unreachable);
    Badge current_badge()
        raises (Domain_Repository::Destination_Unreachable);
    Sensor home_sensor()
        raises (Domain_Repository::Destination_Unreachable);
    ABng_Core::Locatable_Type_Specific_Object type_specific_object()
        raises (Domain_Repository::Destination_Unreachable);

};

#endif                                        /* __BADGE_HOLDER_IDL__ */
```

The *ABng_Location_Description* interface provides information about distribution of sensors among locations.

```
#ifndef __ABNG_LOCATION_DESCRIPTION_IDL__
#define __ABNG_LOCATION_DESCRIPTION_IDL__ 1

interface ABng_Location_Description : System_Static_Data_Observed_E {

    struct Description {
        Domain_Repository::Repository_Item_Id item_id;
        Domain_Repository::Location_Description loc_descr;
        Sensors sensor_list;
    };

    typedef sequence<Description> Descriptions;

    struct Value_Description {
        Domain_Repository::Repository_Item_Id item_id;
        Domain_Repository::Location_Description::Value_Description
            loc_descr;
        Sensor::Value_Descriptions sensor_list;
    };

    typedef sequence<Value_Description> Value_Descriptions;

    struct Pattern {
        boolean is_any;
        Domain_Repository::Repository_Item_Id_Pattern item_id_pattern;
        Domain_Repository::Location_Description::Pattern
            loc_descr_pattern;
        Sensor::Sensors_Pattern sensor_list_pattern;
    };
```

```
    Description descr() raises(Domain_Repository::Destination_Unreachable);
    Value_Description value_descr()
        raises(Domain_Repository::Destination_Unreachable);

    readonly attribute Domain_Repository::Repository_Item_Id item_id;

    Domain_Repository::Location_Description loc_descr()
        raises(Domain_Repository::Destination_Unreachable);
    Sensors sensor_list() raises(Domain_Repository::Destination_Unreachable);

    void add(in Sensor new_sensor)
        raises(Domain_Repository::Duplicate_Data_Record,
            Domain_Repository::Unknown_Object_Ref);
    void remove(in Sensor sensor_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
};

#endif                              /* __ABNG_LOCATION_DESCRIPTION_IDL__ */
```

The *ABng_Domain_System* interface holds a reference of the *Scheduler* and *ABng_Domain* objects.

```
#ifndef __ABNG_DOMAIN_SYSTEM_IDL__
#define __ABNG_DOMAIN_SYSTEM_IDL__ 1

interface ABng_Domain_System {

    readonly attribute ABng_Core::ABng_Domain domain;
    readonly attribute Scheduler my_scheduler;
};

#endif                              /* __ABNG_DOMAIN_SYSTEM_IDL__ */
```

5.4 ABng Internal Interfaces

The *ABng_Core_Internals* module is defined as follows.

```
#ifndef __ABNG_CORE_INTERNALS_IDL__
#define __ABNG_CORE_INTERNALS_IDL__ 1

#include "abng_core_system.idl"

module ABng_Core_Internals {
#include "abng_core_internals_types.idl"
#include "locatable_sighting_processor.idl"
#include "location_sighting_processor.idl"
#include "view_sighting_processor.idl"
#include "main_sighting_processor.idl"
#include "abng_domain_internals.idl"
};

#endif                              /* __ABNG_CORE_INTERNALS_IDL__ */
```

The IDL files included in the above definition are presented and described below. The *abng_core_system_types.idl* file defines the internal representation of a sighting message.

```
#ifndef __ABNG_CORE_INTERNALS_TYPES_IDL__
#define __ABNG_CORE_INTERNALS_TYPES_IDL__ 1

// Internal interfaces
interface ABng_Domain_Internals;
interface View_Sighting_Processor;
interface Main_Sighting_Processor;
interface Locatable_Sighting_Processor;
interface Location_Sighting_Processor;

// An internal representation of sighting message used by sighting
// processors for exchanging sighting data.
struct Internal_Sighting_Description {
    ABng_Core_System::Sensor_Id my_sensor;
    ABng_Core_System::Badge_Id my_badge;
    unsigned short message_type;
    unsigned short attributes;
    ABng_Core::Seq_Number sequence_number;
    ABng_Core::Time update_time;
};

#endif                                   /* __ABNG_CORE_INTERNALS_TYPES_IDL__ */
```

The next four include files define sighting processors. Their role is described in section ?? in details.

```
#ifndef __LOCATABLE_SIGHTING_PROCESSOR_IDL__
#define __LOCATABLE_SIGHTING_PROCESSOR_IDL__ 1

interface Locatable_Sighting_Processor {

    struct Sighting {
        ABng_Core::Location my_location;                    // Current badge location
        ABng_Core_System::Sensor_Id my_sensor;
        unsigned short messsage_type;
        unsigned short click_number;
        ABng_Core::Seq_Number sequence_number;
        ABng_Core::Time update_time;
        ABng_Core::Quality q;
    };

    readonly attribute ABng_Core::Locatable my_locatable;
    readonly attribute ABng_Core_System::Badge_Holder my_badge_holder;
    readonly attribute Sighting current_sighting;

    void process_sighting(
        in Internal_Sighting_Description sighting_descr,
        in ABng_Core::Location loc);
};
```

```
#endif                                  /* __LOCATABLE_SIGHTING_PROCESSOR_IDL__ */

#ifndef __LOCATION_SIGHTING_PROCESSOR_IDL__
#define __LOCATION_SIGHTING_PROCESSOR_IDL__ 1

typedef sequence <Locatable_Sighting_Processor> Locatable_Sighting_Processors;

interface Location_Sighting_Processor {

    enum Event_Descriptor {
        CAME_IN,
        CAME_OUT
    };

    readonly attribute ABng_Core::Location my_location;
    readonly attribute ABng_Core_System::ABng_Location_Description
                my_abng_location_description;
    // A list of locatable sighting processors associated with locatables
    // seen in the location pointed by my_location
    readonly attribute Locatable_Sighting_Processors badge_sp_list;

    // Update a list of seen locatables
    void process_sighting(in Locatable_Sighting_Processor locatable_sp,
        in Event_Descriptor event);

};

#endif                                  /* __LOCATION_SIGHTING_PROCESSOR_IDL__ */

#ifndef __VIEW_SIGHTING_PROCESSOR_IDL__
#define __VIEW_SIGHTING_PROCESSOR_IDL__ 1

//typedef sequence<Location_Sighting_Processor> Location_Sighting_Processors;

interface View_Sighting_Processor {

    readonly attribute ABng_Core::View my_view;
    //readonly attribute Location_Sighting_Processors location_sp_list;
    //readonly attribute Locatable_Sighting_Processors locatable_sp_list;

    readonly attribute ABng_Core_System::Sensor_Ids sensor_id_list;
    readonly attribute ABng_Core_System::Badge_Ids badge_id_list;

    oneway void process_sighting(
        in Internal_Sighting_Description sighting_descr);
};

#endif                                  /* __VIEW_SIGHTING_PROCESSOR_IDL__ */

#ifndef __MAIN_SIGHTING_PROCESSOR_IDL__
#define __MAIN_SIGHTING_PROCESSOR_IDL__ 1
```

```
interface Main_Sighting_Processor {

    typedef sequence<View_Sighting_Processor> View_Sighting_Processors;

    readonly attribute ABng_Domain_Internals my_domain;
    readonly attribute View_Sighting_Processors view_sp_list;

    void add_vsp(in string view_name,
        in ABng_Core_Internals::View_Sighting_Processor vsp,
        in ABng_Core_System::Badge_Ids badge_id_list,
        in ABng_Core_System::Sensor_Ids sensor_id_list,
        in boolean wants_duplicates)
        raises(Domain_Repository::Unknown_Object_Ref,
            Domain_Repository::Duplicate_Data_Record);
    void find_vsp(in string view_name,
        out ABng_Core_Internals::View_Sighting_Processor vsp,
        out ABng_Core_System::Badge_Ids badge_id_list,
        out ABng_Core_System::Sensor_Ids sensor_id_list,
        out boolean wants_duplicates)
        raises(ABng_Core::Unknown_View_Name);
    void remove_vsp(in string view_name)
        raises(ABng_Core::Unknown_View_Name);
    void add_badges(in string view_name,
        in ABng_Core_System::Badge_Ids badge_id_list)
        raises(ABng_Core::Unknown_View_Name);
    void remove_badges(in string view_name ,
        in ABng_Core_System::Badge_Ids badge_id_list)
        raises(ABng_Core::Unknown_View_Name,
            ABng_Core_System::Unknown_Badge_Id);
    void add_sensors(in string view_name,
        in ABng_Core_System::Sensor_Ids sensor_id_list)
        raises(ABng_Core::Unknown_View_Name);
    void remove_sensors(in string view_name,
        in ABng_Core_System::Sensor_Ids sensor_id_list)
        raises(ABng_Core::Unknown_View_Name,
            ABng_Core_System::Unknown_Sensor_Id);
};

#endif                              /* __MAIN_SIGHTING_PROCESSOR_IDL__ */
```

The *ABng_Domain_Internals* interface holds a reference of the main sighting processor and system domain.

```
#ifndef __ABNG_DOMAIN_INTERNALS_IDL__
#define __ABNG_DOMAIN_INTERNALS_IDL__ 1

interface ABng_Domain_Internals {

    readonly attribute ABng_Core_System::ABng_Domain_System domain;
    readonly attribute Main_Sighting_Processor main_sp;
};
```

```
#endif                                   /* __ABNG_DOMAIN_INTERNALS_IDL__ */
```

5.5 ABng Management Interfaces

The *ABng_Core_Management* module is defined as follows.

```
#include <orb.idl>

#include "abng_core_internals.idl"

module ABng_Core_Management {

interface Badge_Rep ;
interface Badge_Holder_Rep ;
interface Sensor_Rep ;
interface ABng_Location_Description_Rep ;
interface View_Manager ;
interface View_Rep_Manager ;

interface Badge_Rep : ABng_Core_System :: System_Static_Data_Observed_R {

readonly attribute ABng_Core_System :: Badges badge_list ;

ABng_Core_System :: Badge add (
in ABng_Core_System :: Badge :: Description data_record )
raises ( Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Destination_Unreachable ) ;
void remove ( in ABng_Core_System :: Badge object_to_remove )
raises ( Domain_Repository :: Unknown_Object_Ref ) ;
ABng_Core_System :: Badges find (
in ABng_Core_System :: Badge :: Pattern data_record_pattern )
raises ( Domain_Repository :: Search_Not_Implemented ) ;
ABng_Core_System :: Badge :: Value_Descriptions find_values (
in ABng_Core_System :: Badge :: Pattern data_record_pattern )
raises ( Domain_Repository :: Search_Not_Implemented ) ;
void update ( in ABng_Core_System :: Badge object_to_update ,
in ABng_Core_System :: Badge :: Description data_record_pattern )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Destination_Unreachable ) ;
} ;

interface Badge_Holder_Rep : ABng_Core_System :: System_Static_Data_Observed_R {

typedef sequence < ABng_Core_System :: Badge_Holder > Badge_Holders ;

readonly attribute Badge_Holders badge_holder_list ;

ABng_Core_System :: Badge_Holder add (
in ABng_Core_System :: Badge_Holder :: Description data_record )
raises ( Domain_Repository :: Duplicate_Data_Record ,
```

```
Domain_Repository :: Destination_Unreachable ) ;
void remove ( in ABng_Core_System :: Badge_Holder object_to_remove )
raises ( Domain_Repository :: Unknown_Object_Ref ) ;
Badge_Holders find (
in ABng_Core_System :: Badge_Holder :: Pattern data_record_pattern )
raises ( Domain_Repository :: Search_Not_Implemented ,
Domain_Repository :: Destination_Unreachable ) ;
ABng_Core_System :: Badge_Holder :: Value_Descriptions find_values (
in ABng_Core_System :: Badge_Holder :: Pattern data_record_pattern )
raises ( Domain_Repository :: Destination_Unreachable ,
Domain_Repository :: Search_Not_Implemented ) ;
void update ( in ABng_Core_System :: Badge_Holder object_to_update ,
in ABng_Core_System :: Badge_Holder :: Description data_record_pattern )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Destination_Unreachable ) ;
} ;

interface Sensor_Rep : ABng_Core_System :: System_Static_Data_Observed_R {

typedef ABng_Core_System :: Sensor :: Sensors_Pattern Sensors_Pattern ;

readonly attribute ABng_Core_System :: Sensors sensor_list ;

ABng_Core_System :: Sensor add (
in ABng_Core_System :: Sensor :: Description data_record )
raises ( Domain_Repository :: Duplicate_Data_Record ) ;
void remove ( in ABng_Core_System :: Sensor object_to_remove )
raises ( Domain_Repository :: Unknown_Object_Ref ) ;
ABng_Core_System :: Sensors find (
in ABng_Core_System :: Sensor :: Pattern data_record_pattern )
raises ( Domain_Repository :: Search_Not_Implemented ) ;
ABng_Core_System :: Sensor :: Value_Descriptions find_values (
in ABng_Core_System :: Sensor :: Pattern data_record_pattern )
raises ( Domain_Repository :: Search_Not_Implemented ) ;
void update ( in ABng_Core_System :: Sensor object_to_update ,
in ABng_Core_System :: Sensor :: Description data_record_pattern )
raises ( Domain_Repository :: Unknown_Object_Ref ,
Domain_Repository :: Duplicate_Data_Record ) ;
} ;

interface ABng_Location_Description_Rep :
ABng_Core_System :: System_Static_Data_Observed_R {

readonly attribute ABng_Core_System :: ABng_Location_Descriptions
abng_location_descriptions_list ;

ABng_Core_System :: ABng_Location_Description add (
in ABng_Core_System :: ABng_Location_Description :: Description
data_record )
raises ( Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Invalid_Description ) ;
```

void remove (**in** ABng_Core_System :: ABng_Location_Description
object_to_remove)
raises (Domain_Repository :: Unknown_Object_Ref) ;
ABng_Core_System :: ABng_Location_Descriptions find (
in ABng_Core_System :: ABng_Location_Description :: Pattern
data_record_pattern)
raises (Domain_Repository :: Search_Not_Implemented ,
Domain_Repository :: Destination_Unreachable) ;
ABng_Core_System :: ABng_Location_Description :: Value_Descriptions
find_values (
in ABng_Core_System :: ABng_Location_Description :: Pattern
data_record_pattern)
raises (Domain_Repository :: Search_Not_Implemented ,
Domain_Repository :: Destination_Unreachable) ;
void update (**in** ABng_Core_System :: ABng_Location_Description
object_to_update ,
in ABng_Core_System :: ABng_Location_Description :: Description
data_record_pattern)
raises (Domain_Repository :: Unknown_Object_Ref ,
Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Invalid_Description) ;
} ;

interface View_Manager {

exception Invalid_Location_Type { } ;
readonly attribute ABng_Core :: View my_view ;

ABng_Core :: Location add_location (
in ABng_Core_System :: ABng_Location_Description new_data)
raises (Invalid_Location_Type ,
Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Destination_Unreachable) ;

void remove_location (
in ABng_Core :: Location object_to_remove)
raises (Domain_Repository :: Unknown_Object_Ref) ;

ABng_Core :: Locatable add_locatable (
in ABng_Core_System :: Badge_Holder new_data)
raises (Domain_Repository :: Duplicate_Data_Record ,
Domain_Repository :: Destination_Unreachable ,
Domain_Repository :: Invalid_Badge_Holder) ;

void remove_locatable (
in ABng_Core :: Locatable object_to_remove)
raises (Domain_Repository :: Unknown_Object_Ref) ;
} ;

interface View_Rep_Manager {

```
typedef sequence < View_Manager > View_Managers ;

readonly attribute ABng_Core :: View_Rep rep ;

readonly attribute View_Managers view_manager_list ;

View_Manager add ( in string view_name )
raises ( Domain_Repository :: Duplicate_Data_Record ) ;
void remove ( in View_Manager object_to_remove )
raises ( Domain_Repository :: Unknown_Object_Ref ) ;
View_Managers find ( in string name )
raises ( Domain_Repository :: Search_Not_Implemented ) ;
} ;

interface ABng_Domain_Management {

readonly attribute ABng_Core_Internals :: ABng_Domain_Internals domain ;
readonly attribute Badge_Rep my_badge_rep ;
readonly attribute Badge_Holder_Rep my_badge_holder_rep ;
readonly attribute Sensor_Rep my_sensor_rep ;
readonly attribute ABng_Location_Description_Rep my_abng_loc_descr_rep ;
readonly attribute View_Rep_Manager my_view_rep_manager ;

} ;

} ;

#ifndef __ABNG_CORE_MANAGEMENT_TYPES_IDL__
#define __ABNG_CORE_MANAGEMENT_TYPES_IDL__ 1

interface Badge_Rep;
interface Badge_Holder_Rep;
interface Sensor_Rep;
interface ABng_Location_Description_Rep;
interface View_Manager;
interface View_Rep_Manager;

#endif                          /* __ABNG_CORE_MANAGEMENT_TYPES_IDL__ */
```

In the module interfaces of badge, badge holder, sensor and location description repositories are defined. (Note, that operations for searching and updating have not been separated as it has been done in a case of domain database repositories. This is because badge, badge holder, sensor and location description repositories are intended to be used only by system managers.)

```
#ifndef __BADGE_REP_IDL__
#define __BADGE_REP_IDL__ 1

interface Badge_Rep : ABng_Core_System::System_Static_Data_Observed_R {

    readonly attribute ABng_Core_System::Badges badge_list;

    ABng_Core_System::Badge add(
```

```
        in ABng_Core_System::Badge::Description data_record)
        raises(Domain_Repository::Duplicate_Data_Record,
                Domain_Repository::Destination_Unreachable);
    void remove(in ABng_Core_System::Badge object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
    ABng_Core_System::Badges find(
        in ABng_Core_System::Badge::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented);
    ABng_Core_System::Badge::Value_Descriptions find_values(
        in ABng_Core_System::Badge::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented);
    void update(in ABng_Core_System::Badge object_to_update,
        in ABng_Core_System::Badge::Description data_record_pattern)
        raises(Domain_Repository::Unknown_Object_Ref,
            Domain_Repository::Duplicate_Data_Record,
            Domain_Repository::Destination_Unreachable);
};

#endif                                          /* __BADGE_REP_IDL__ */

#ifndef __BADGE_HOLDER_REP_IDL__
#define __BADGE_HOLDER_REP_IDL__ 1

interface Badge_Holder_Rep : ABng_Core_System::System_Static_Data_Observed_R {

    typedef sequence<ABng_Core_System::Badge_Holder> Badge_Holders;

    readonly attribute Badge_Holders badge_holder_list;

    ABng_Core_System:: Badge_Holder add(
        in ABng_Core_System::Badge_Holder::Description data_record)
        raises(Domain_Repository::Duplicate_Data_Record,
                Domain_Repository::Destination_Unreachable);
    void remove(in ABng_Core_System::Badge_Holder object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
    Badge_Holders find(
        in ABng_Core_System::Badge_Holder::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented,
                Domain_Repository::Destination_Unreachable);
    ABng_Core_System::Badge_Holder::Value_Descriptions find_values(
        in ABng_Core_System::Badge_Holder::Pattern data_record_pattern)
        raises(Domain_Repository::Destination_Unreachable,
                Domain_Repository::Search_Not_Implemented);
    void update(in ABng_Core_System::Badge_Holder object_to_update,
        in ABng_Core_System::Badge_Holder::Description data_record_pattern)
        raises(Domain_Repository::Unknown_Object_Ref,
                Domain_Repository::Duplicate_Data_Record,
                Domain_Repository::Destination_Unreachable);
};

#endif                                   /* __BADGE_HOLDER_REP_IDL__ */

#ifndef __SENSOR_REP_IDL__
```

```
#define __SENSOR_REP_IDL__ 1

interface Sensor_Rep : ABng_Core_System::System_Static_Data_Observed_R {

    typedef ABng_Core_System::Sensor::Sensors_Pattern Sensors_Pattern;

    readonly attribute ABng_Core_System::Sensors sensor_list;

    ABng_Core_System::Sensor add(
        in ABng_Core_System::Sensor::Description data_record)
        raises(Domain_Repository::Duplicate_Data_Record);
    void remove(in ABng_Core_System::Sensor object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
    ABng_Core_System::Sensors find(
        in ABng_Core_System::Sensor::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented);
    ABng_Core_System::Sensor::Value_Descriptions find_values(
        in ABng_Core_System::Sensor::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented);
    void update(in ABng_Core_System::Sensor object_to_update,
        in ABng_Core_System::Sensor::Description data_record_pattern)
        raises(Domain_Repository::Unknown_Object_Ref,
            Domain_Repository::Duplicate_Data_Record);
};

#endif                                          /* __SENSOR_REP_IDL__ */

#ifndef __ABNG_LOCATION_DESCRIPTION_REP_IDL__
#define __ABNG_LOCATION_DESCRIPTION_REP_IDL__ 1

interface ABng_Location_Description_Rep :
    ABng_Core_System::System_Static_Data_Observed_R {

    readonly attribute ABng_Core_System::ABng_Location_Descriptions
        abng_location_descriptions_list;

    ABng_Core_System::ABng_Location_Description add(
        in ABng_Core_System::ABng_Location_Description::Description
            data_record)
        raises(Domain_Repository::Duplicate_Data_Record,
               Domain_Repository::Invalid_Description);
    void remove(in ABng_Core_System::ABng_Location_Description
            object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
    ABng_Core_System::ABng_Location_Descriptions find(
        in ABng_Core_System::ABng_Location_Description::Pattern
            data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented,
               Domain_Repository::Destination_Unreachable);
    ABng_Core_System::ABng_Location_Description::Value_Descriptions
        find_values(
        in ABng_Core_System::ABng_Location_Description::Pattern
            data_record_pattern)
```

```
        raises(Domain_Repository::Search_Not_Implemented,
               Domain_Repository::Destination_Unreachable);
    void update(in ABng_Core_System::ABng_Location_Description
            object_to_update,
        in ABng_Core_System::ABng_Location_Description::Description
            data_record_pattern)
        raises(Domain_Repository::Unknown_Object_Ref,
               Domain_Repository::Duplicate_Data_Record,
               Domain_Repository::Invalid_Description);
};

#endif                          /* __ABNG_LOCATION_DESCRIPTION_REP_IDL__ */
```

The *ABng_Core_Management* module also contains the interface used to manage views.

```
#ifndef __VIEW_MANAGER_IDL__
#define __VIEW_MANAGER_IDL__ 1

interface View_Manager {

    exception Invalid_Location_Type {};
    readonly attribute ABng_Core::View my_view;

    // Add a new location for a given ABng location description
    ABng_Core::Location add_location(
        in ABng_Core_System::ABng_Location_Description new_data)
        raises(Invalid_Location_Type,
            Domain_Repository::Duplicate_Data_Record,
            Domain_Repository::Destination_Unreachable);
    // Remove a location for a given ABng location description
    void remove_location(
        in ABng_Core::Location object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
    // Add a new locatable for a given badge holder
    ABng_Core::Locatable add_locatable(
        in ABng_Core_System::Badge_Holder new_data)
        raises(Domain_Repository::Duplicate_Data_Record,
            Domain_Repository::Destination_Unreachable,
            Domain_Repository::Invalid_Badge_Holder);
    // Remove a location for a given badge holder
    void remove_locatable(
        in ABng_Core::Locatable object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
};

#endif                                    /* __VIEW_MANAGER_IDL__ */

#ifndef __VIEW_REP_MANAGER_IDL__
```

```
#define __VIEW_REP_MANAGER_IDL__ 1

interface View_Rep_Manager {

    typedef sequence<View_Manager> View_Managers;

    readonly attribute ABng_Core::View_Rep rep;
    // list of all view managers:
    readonly attribute View_Managers view_manager_list;

    View_Manager add(in string view_name)
        raises(Domain_Repository::Duplicate_Data_Record);
    void remove(in View_Manager object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);
    View_Managers find(in string name)
        raises(Domain_Repository::Search_Not_Implemented);
};

#endif                                  /* __VIEW_REP_MANAGER_IDL__ */
```

References of all repository objects, view management object and internal domain are hold in the *ABng_Domain_Management* interface.

```
#ifndef __ABNG_DOMAIN_MANAGEMENT_IDL__
#define __ABNG_DOMAIN_MANAGEMENT_IDL__ 1

interface ABng_Domain_Management {

    readonly attribute ABng_Core_Internals::ABng_Domain_Internals domain;
    readonly attribute Badge_Rep my_badge_rep;
    readonly attribute Badge_Holder_Rep my_badge_holder_rep;
    readonly attribute Sensor_Rep my_sensor_rep;
    readonly attribute ABng_Location_Description_Rep my_abng_loc_descr_rep;
    readonly attribute View_Rep_Manager my_view_rep_manager;

};

#endif                                  /* __ABNG_DOMAIN_SYSTEM_IDL__ */
```

6 Sighting Processing

The sighting processing has the following steps depicted in the Figure 2 and commented below.

1. A poller invokes the *sighting* operation of the Main Sighting Processor (MSP) (arrow 1 in Figure 2). MSP provides the interface (*Main_Sighting_Processor*) which is derived from *AB_Observer*. The MSP is registered within the poller. The MSP receives all data obtained from the badge message and the poller (group) name.

2. The MSP has an index table which maps badge identifiers into a structure that contains:

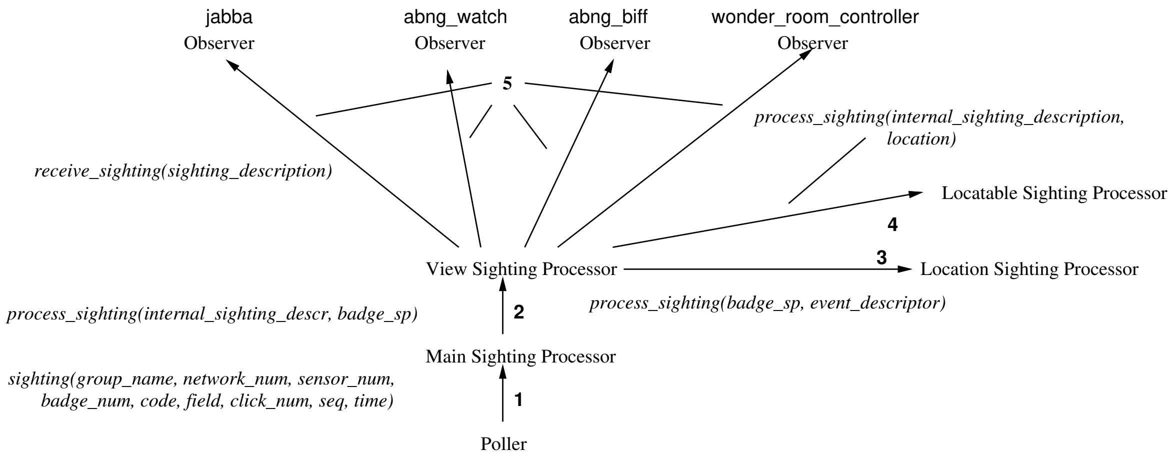

Figure 2: Sighting Processing Path

- a sequence number of the last sighting message emitted by the badge (this number is initialized to 0);
- a list of View Sighting Processors (VSPs) associated with views to which the given badge belongs.

The MSP compares a sequence number of the message obtained from the poller to a sequence number stored in the table. If they are the same, that means that the message is a copy of the message processed previously for the given badge. In that case the message is discarded.

Otherwise, the MSP creates an *Internal_Sighting_Description* structure containing all the information obtained from the poller and sends it to all View Sighting Processors (from the list associated with the badge which emitted the sighting) via invocation of the the *process_sighting* operation of each VSP (arrow 2). Note that the *process_sighting* method is one-way and does not suspend the MSP which may process the next sighting after it has sent the previous one to all interested VSPs.

3. A VSP manages two lists:

- a list of Location Sighting Processors (LSP) associated with locations that belong to the view;
- a list of LocatablE Sighting Processors (LESP) corresponding to locatables (and in a consequence to badges) belonging to the view.

Additionally, a VSP holds a list of observers of each locatable and location. These lists are updated by applications vis the *View*, *Location* and *Locatable* interfaces.

The VSP also maintains two index tables. One of them maps full sensor identifiers into LSPs associated with location that contain given sensors. Another one maps badge identifiers into LESP references.

Moreover, for each badge the VSP has a cache that contains an identifier of the sensor that received the last badge message. (This sensor will be called *old sensor* whereas a sensor identifier included in the internal sighting description will be called *new sensor*.)

Using the index table VSP determines what kind of location change has occurred and which locations have been affected by this change. The VSP may recognize one of the following situations:

(a) The badge changed a location within the view – both the new and the old location belong to the view. This is the case when both old and new sensor identifiers are different and belong to two distinct locations of the view.

(b) The badge entered the view – the new location belongs to the view but the old one does not.

(c) The badge came out of the view – the old location belongs to the view but the new one does not.

(d) The badge did not change the location.

4. In case a) the VSP informs the LSP of the old location that the given badge left a location. In case a) or b) the VSP informs the LSP of the new location that the given badge entered the location (arrow 3). The VSP passes a reference of the LESP and event type (entering or leaving) to the LSP.

   An LSP has a list of LESPs associated with badges seen in the LSP's location. The LSP adds a LESP to the list if the badge entered the location or removes a LESP if the badge left the location.

5. In case a), b) and d) the VSP passes the internal sighting description and the LSP of the current location to the LESP (arrow 4). Using this data the LESP sets the value of its *current_sighting* attribute which is then used to built a sighting description passed to the application.

6. After the location data maintained by the LESP is updated the information about the badge location within the view is complete and consistent. This is the moment when a VSP triggers applications that observe the old or new or both locations or the badge. A list of observers, to which a sighting is sent, is constructed as a concatenation of observer lists associated with locations (one or two) and a locatable affected by the sighting. Duplicated observer references (a given observer may be registered in many locations and locatables) are removed from the list in order to ensure that only one copy of the sighting will be send to each observer.

   To notify observers, a VSP constructs the *Sighting_Description* structure that contains all sighting data. (In order to obtain those data the LSP has to contact the LESP.) Then the VSP calls the *receive_sighting* operation of all observing applications.

   Since the *receive_sighting* method is one-way it does not suspend a VSP. However, because the number of observers may be large, notification of all of them by a separate one-way invocation may be time-consuming. To reduce this time CORBA Event Service will be used as an alternative mechanism of observer notification.

7 Notification Model

There are two types of data used and managed by ABng: dynamic and static data. Dynamic data comprise location information which may change quite frequently (a few times per minute for a single locatable object). Static data characterize logical and physical components of the system, such as badges, sensors, users, equipment, locations, location types, view, etc. They are updated by system managers. This happens relatively rarely.
Furthermore, static data may be divided into:

- public data available for ABng applications (information about views, locations, locatables, users, equipment, etc.)
- system data available for system managers (information about badges, sensors, etc.)

ABng data are encapsulated by CORBA objects providing interfaces described in section **??**. The location data may be obtained using the *View* interface calling the *get_sightings* method which returns a list of sighting descriptions for locatables belonging to the view. Additionally, an application may retrieve a list of location or locatable objects and enquire about location information of some of them using the *Location* or *Locatable* interface.

Static data may be obtained using interfaces of objects representing particular system components, e.g. *User*, *Location*, *Badge*, etc. Such objects will be further called *atomic objects* in contrast to *collection objects* that are collections of atomic objects, e.g. the *User_Rep* or the *View* objects.

An application may retrieve a single attribute of an atomic object (e.g. a user name) by invocation of the appropriate attribute access method. Each object has a special attribute, called *description*, which is a collection of all remaining attributes. Thus a client may obtain all object data in a single invocation.

Atomic objects are related to each other, for instance a location is related to its location description, a locatable is related to a user or a piece of equipment. Such relationships are represented by attributes of a reference type. To obtain attributes of related objects, an application has to use reference attributes that point to the related objects and invoke appropriate access methods on these references. Hence, gathering information about all related objects (e.g. a location, location description and location type) may require a few remote invocations. In some cases this may be inefficient and inconvenient. To solve this problem, objects representing system components has the *get_value_description* method that returns the description of the given object, as well as descriptions of related objects, by value. This method allows to reduce communication overhead because the attribute values of the related objects are cached in the collection object which contains the given atomic object. A mechanism for cache updating is described below in this section.

To access an object attribute an application has to posses the object reference. Location and locatable references may be retrieved from a view object. View references are stored in the view repository. Having locatable and location references, an application can get their attributes and, as a consequence, references of related objects, which attributes may be retrieved next. This is the only way, an application may access database object attributes. ABng applications are not supposed to use database repository interfaces to obtain object references.

Getting location and static data on demand is not sufficient for some applications which need asynchronous notification about location data changes and database updates. In ABng the observed/observer paradigm is employed for client notification. An application has to implement an observer interface which has to be registered within an observed object. In ABng there is a wide range of observable objects of different granularity: there are atomic objects which may be observed as well as collection objects. An observer of the atomic object is notified about a change of any attribute of an object (this attribute may contain static or dynamic data) and about object removal (this concern an observer of static data only). Apart from these events, en observer of a collection object is notified when a new object is added to the collection.

Maintaining an observer registry for each observable object individually would be an inefficient solution. Therefore, an observer registry is maintained by a collection object only. An atomic object does not have its own observer registry. When a client registers an observer within a given atomic object, this registration is forwarded to a collection object which updates the observer registry.

When an application registers or removes an observer within a collection object it can specify a list of atomic objects belonging to the collection in which the application is interested. (Note that the same interface is used for registration in atomic and collection objects and the list parameter is ingored in a case of atomic object.) If an empty list is given an observer is registered in all objects of a collection. It should be noted that registration via a collection object is usually faster that direct registration in atomic object.

In general, within the application many observers of the same object may exist. Registering all of them in the atomic object and storing all their references in a collection object would introduce unnecessary overheads. Instead, in the application the notification dispatcher is created which provides the observer interface. This dispatcher is registered in the collection object and it is a representant of all remaining observers in the application within the given collection. It dispatches notification messages to represented observers.

In Orbix or OrbixWeb applications, an existence of the notification dispatcher may be hidden thanks to the smart proxy mechanism. Moreover, smart proxies may used for caching attribute values of atomic objects transparently for the application.

Below, a typical scenario of the ABng application behaviour is described. It illustrates how smart proxies and notification dispatchers will be used. Some of the scenario phases are depicted in Figure 3.

1. An application contacts the *View_Rep* and finds a *View* object using the view name as a key.

2. The application gets lists of *Location* and *Locatable* objects. When an object reference comes in the application process, a smart proxy object is created.

3. The application creates an instance of the *View_Data_Observer* interface and registers it in all *Location* and *Locatable* objects calling the *attach_view_data_observer* method of the

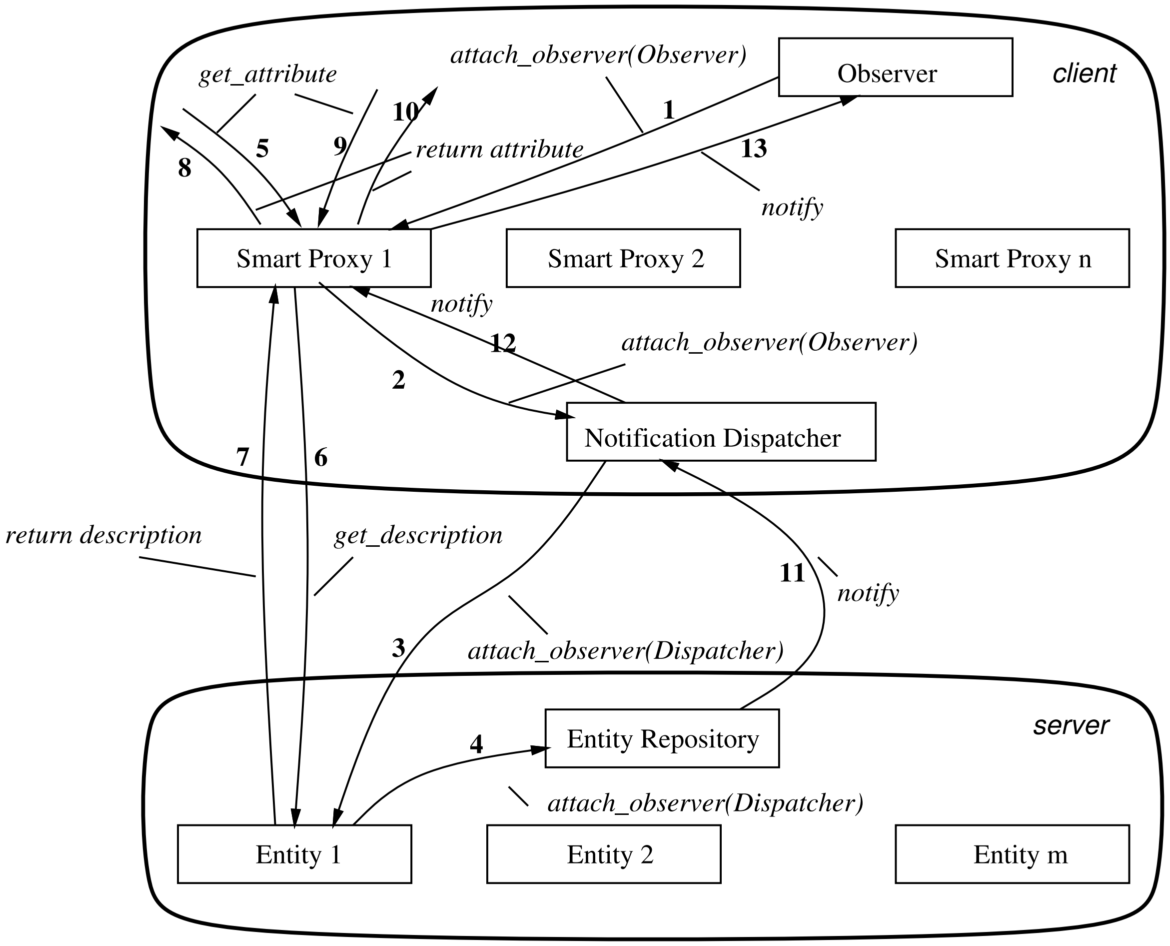

Figure 3: The role of smart proxies in caching and notification

View object. This ensures that the application will be notified about changes of attributes defined in the *Location* and *Locatable* interface, for instance, about a change of an object home location, or a change of a reference to the *Location_Description* object. (Registration in many *Location* and *Locatable* objects should be performed via the single call of the *View* interface rather than many invocations of atomic interfaces, because it is more efficient.)

4. The application calls the access method of the *description* attribute of each *Location* and *Locatable*. Since*Location* and *Locatable* contain references of *Location_Description* and *User* or *Equipment* from domain databases, for each such a reference a smart proxy object is created within the application.

5. If the application wants to monitor changes attributes of database objects related to obtained *Location* and *Locatable*, it has to register itself in these objects. Let us assume that in our scenario, the application wants to monitor changes of user attributes. To do it, the application creates an instance of the *Static_Data_Observer* interface and calls the *attach_static_data_observer* of each *User* object. In fact, the application calls the method

of the smart proxy for the *User* object (arrow 1 in Fig. 3).

6. The smart proxy inserts a reference of the observer (a C++ or Java pointer to observer implementation) into the list that contains all references of all observers of the given atomic object within the application process. After that, the smart proxy calls the notification dispatcher (arrow 2).

7. The dispatcher casts the smart proxy pointer to the ordinary proxy pointer and calls the *attach_static_data_observer* on it. As the argument it passes its own reference. This call results in real remote invocation on the *User* object (arrow 3).

8. The *User* object forwards the invocation to the *User_Repository* implementation object using the internal (C++) interface (arrow 3). The repository adds the reference of the notification dispatcher to the list of observers of the *User* object.

9. The above four steps are performed for each *User* object which the application wants to observe.

10. Assume that after registration, the application gets names of all users. To do it, the application calls the access method of the *surname* attribute of a smart proxy for each *User* (arrow 5).

11. Afterwards, the smart proxy calls the access method of the *description* attribute (arrow 6) and gets the user description that contains all attributes (arrow 7). The description is cached by the proxy.

12. The proxy returns to the caller the name obtained from the description (arrow 8).

13. After that, the application may want to retrieve another attribute, for instance a first-name. Hence, it calls the access method of the given attribute on the proxy (arrow 9). Since, the proxy has already obtained values of all attributes, it simply returns the attribute value (arrow 10).

14. Let us assume that some values of one of the *User* objects change. This may be done only via the *update* method of the *User_Rep* interface. This method uses the observer registry and finds and notifies all observers of the updated object (arrow 11). One of them is the notification dispatcher in our application.

15. The dispatcher obtains a reference of the updated object which is, in fact, a pointer to the object proxy. The dispatcher narrows this pointer to the pointer of the smart proxy and forwards notification to it (arrow 12).

16. The smart proxy, which has a list of object observers in the given process, notifies all of them (arrow 13). In addition, the proxy marks its cache as invalid. The next access to any object attribute will cause the proxy to retrieve current values of all object attributes.

For Orbix and OrbixWeb a library containing smart proxies for all ABng Core and database atomic objects as well as notification dispatchers will be developed. For the remaining CORBA implementations a similar library containing caches and notification dispatchers can be also implemented. However, it is not likely to employ smart proxies, because this is an Orbix-specific mechanism. As a consequence, caching and notification dispatching will not be transparent for an application code and a programmer will have to be aware of their existence.

The notification mechanism, presented above, is also used inside the ABng core and the domain database for notification of a collection object about changes in other related collections. For instance, when a *Badge* object is removed for *Badge_Rep* the *Badge_Holder_Rep* object should be informed about it and if a *Badge_Holder* object is removed the *Badge_Rep* object should be notified. The action performed by the given collection object is specific for this collection and its atomic objects. Typically, the notified collection removes a related object. This is a case for the *Badge_Holder_Rep* which removes *Badge_Holder* for a removed *Badge*. In contrast, *Badge_Rep* does not remove *Badge* of the removed *Badge_Holder*. Instead, it sets the *Badge_Holder* reference as a null value, which means that a given badge does not belong to anybody.

Figure 4 shows a notification dependence graph for ABng and domain database components.

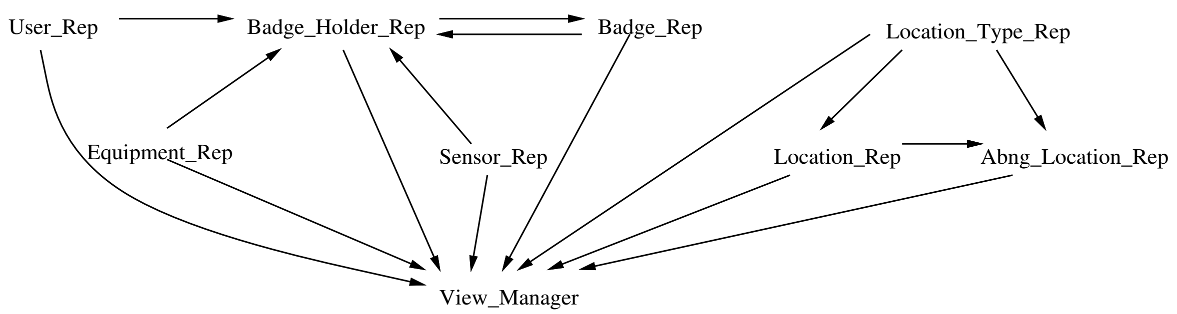

Figure 4: Notification dependence graph for ABng and domain database components

8 Command Execution

Active badges and sensors can execute commands which are sent to them by a poller. The poller–sensor protocol as well as the sensor–badge infra-red protocol do not ensure a reliable command and reply delivery. This implies a need of existence of the module which would buffer commands, wait for replies, retry commands if necessary and forward reply messages to command callers. In ABng this role is performed by the scheduler.

The scheduler interface, presented in section **??**, provides a separate method for each command type. It also has the *issue_command* method which allows to send any message to a sensor or a badge.

After an invocation of a command method, the scheduler assigns a request identifier and stores method arguments in the request list. A caller of a command method may pass more than one sensor or a badge to the scheduler. Moreover, a caller may require to repeat a command for a set of sensors or badges many times. Hence, a single caller request may be related to many command executions. When all commands related to that request are completed, the scheduler removes the request from the list. A list of requests may be searched via the *Scheduler_Request_Queue* base interface. It also enable to remove a request from the list. In that case, all pending commands related to the request are discarded.

Conditions that constrain the way request commands are executed are specified in the context parameter of each command method. These conditions include:

- *at* – the time (number of seconds and microseconds since 1st January 1970, 0:00 am) when a command will be sent out by the scheduler. A command will wait in the waiting command queue until this time. If a command is to be executed as soon as possible, the time should contains zero values.
- *waiting_for_sighting_timout* – the time the scheduler waits for a sighting from the badge to which that the command is addressed (for badge commands only). The badge can receive a command for three seconds after it emits a sighting message. Therefore, the scheduler does not sent a badge command immediately after it obtain a command from a caller or from the waiting command queue. Instead, it waits until a badge sends a sighting. This context attribute specifies the maximum waiting time. After this time, a command is discarded and an error is reported to a caller providing the caller has required error reporting by appropriate setting of the *interest* attribute (see below).
- *waiting_for_reply_timeout* – the maximum time the scheduler waits for a command reply (this is relevant only for commands which have replies). After that time a command is retried (see below) or an error is reported to the caller.
- *retries* – the maximum number of command retries (this is relevant only for commands which have replies). When the scheduler does not receive a command for the *waiting_for_reply_timeout* time, the scheduler sends the command again. After the last retry an error message is reported to the caller.
- *repeat* – the number of command repetitions.
- *interval* – the interval time between subsequent command repetitions.
- *interest* – the mode of notification of a caller.

Below, there is a scheme of an algorithm performed by the scheduler during an execution of a command for a single sensor or a badge.

```
if(at ≠ Time(0,0)) {
    // wait until at
}
for(unsigned i==0; i < repeat; i++) {
  for(unsigned j=0; j < retries; j++) {
      if(                                        /* this is a badge command */) {
          // wait until badge sighting arrives
          // or wating_for_sighting_timout expires ;
          if(                                     /* sighting has not arrived */) {
```

```
                if(interest == Error ||
                    interest == Error_And_Reply ||
                    interest == Error_And_Request_Invoked) {
                    // report error to the caller
                }
                break;                                                              //
            }
        }
        // send command
        if(                                          /* command has no reply */) {
            if(interest == Request_Invoked ||
                interest == Error_And_Request_Invoked {
                // report success to the caller
            }
            break;
        }
        // wait until reply arrives or wating_for_reply_timeout expires
        if(                                                 /* reply has arrived */) {
            if(interest == Reply || interest == Error_And_Reply) {
                // send reply to the caller
            }
            else if(interest == Request_Invoked ||
                interest == Error_And_Request_Invoked) {
                // send success to the caller
            }
            break;
        }
        else {
            // report error to the caller
        }
    }
    // wait for interval;
}
```

If a caller specifies more than one sensor (for sensor commands) or more than one badge (for badge commands) the above algorithm is executed for each badge or a sensor.

Scheduler is a system interface and is not accessible to ABng applications. Instead, they use the *Location_Commander*, *Locatable_Commander* and *Commander* interfaces which are the base interfaces for *Location*, *Locatable* and *View* respectively. Implementations of the first two interfaces simply forward method invocations to *Commander* adding a list of locations or locatables to the argument lists (This forward invocations are made within the same address space because a view and its locations and locatables share the same process.) A view maps lists of locations and locatables into lists of sensors and badges respectively.

The view implementation object implements also the *Scheduler_Caller* interface (the TIE approach is used for that purpose). If a caller has passed a reference of a *Caller* object, the commander replaces it by the reference of the *Scheduler_Caller* CORBA object. When the scheduler completes any command related to the request, it sends a reply or reports and error or success to the view rather than to the original command caller. The view replaces sensor and badge by a location and a locatable (associated with the given badge and sensor) and forwards an invocation to the original caller using its*Caller* interface.

Other command interfaces are *Badge_Commander* and *Sensor_Commander*, which may be only used by manager applications. They simply forward their invocations to the scheduler.

9 Concurrency Model

In some places of sighting processing one-way invocations are used. This causes some actions to be performed concurently. It should be noted that according to our experiments in the current Orbix implementation one-way invocations do not change an order of requests, thus usage of one-way methods does not change an order of sightings.

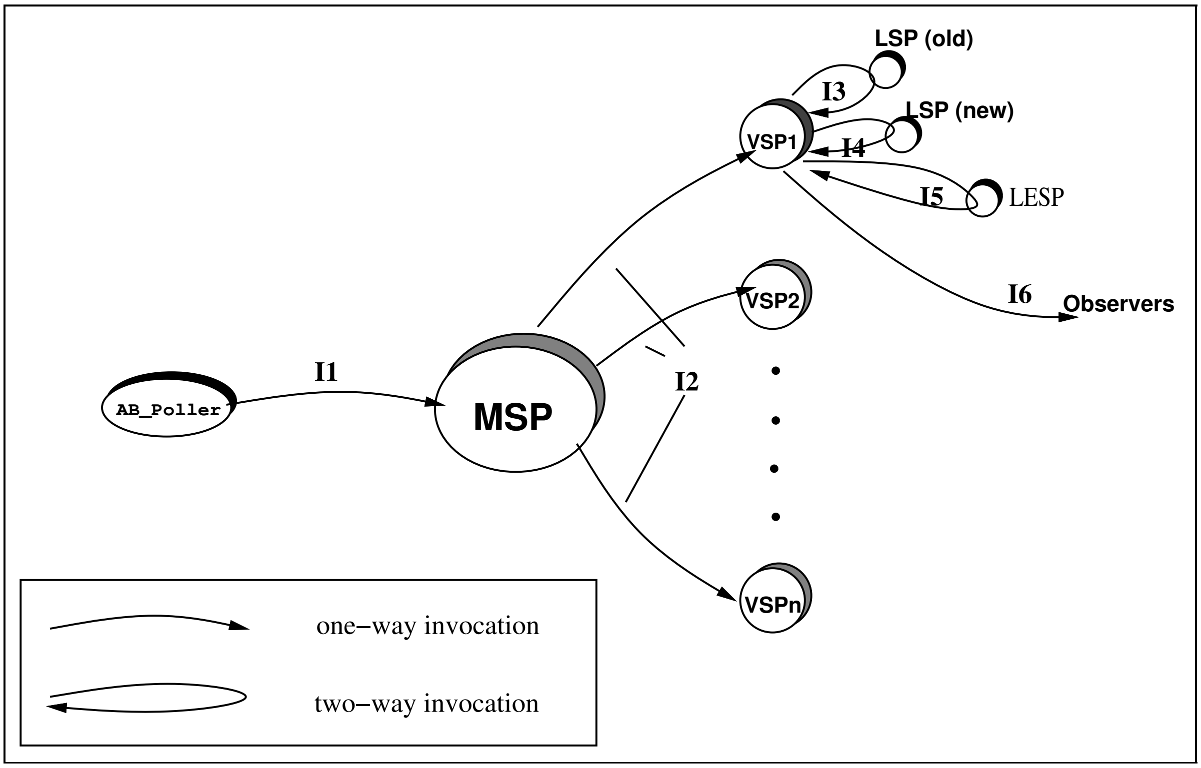

Figure 5: ABng Core Concurrency Model

The following invocations in processing of a sighting are one-way (Fig. 5):

1. Invocations *I1* of the *sighting* method of the MSP made by a poller.
2. Invocations *I2* of the *process_sighting* method of a VSP made by the MSP.
3. Invocations *I6* of the *receive_sighting* method of observers made by a LESP, LSP and VSP.

In the result a single sighting is processed concurrently in particular views andes observers.

10 Implementation Details

Grouping of CORBA objects implementing the interfaces described above in implementation servers is presented in Figure 6.

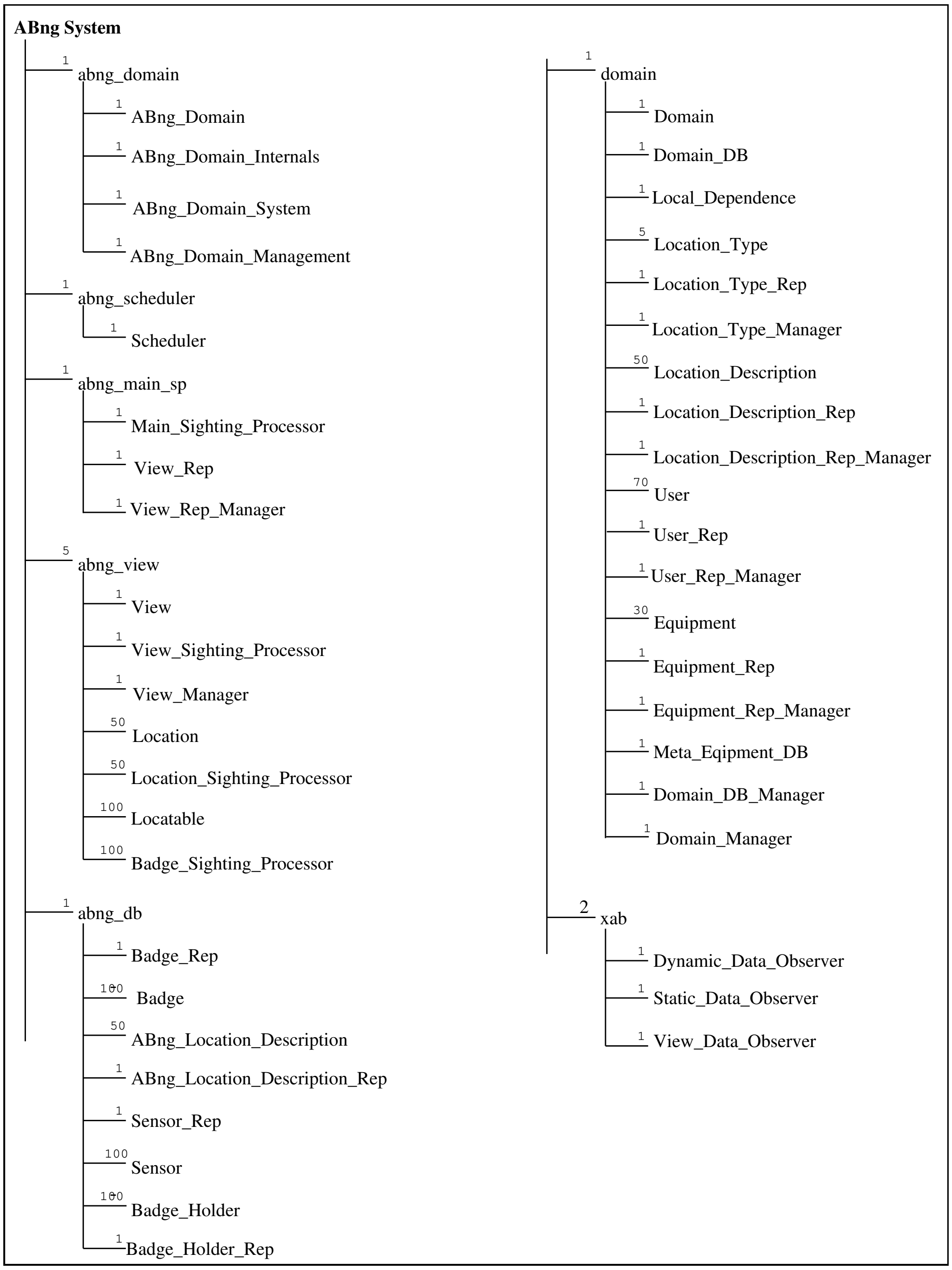

Figure 6: Mapping of ABng object interfaces into implementing server processes

The picture shows two-level tree. The first level are processes and the second one are CORBA interfaces instantiated within those processes. As it has been described previously, the ABng interfaces may be divided into external (public), internal (private for the Core modules), system and management interfaces. Some of the real objects of the system posses several interfaces, e.g. an internal, external and management interface. A state of such objects is modified via an internal or management interface and is provided to user applications vie the public interface. For such objects of the real system the internal, public and management interfaces are implemented by the single implementation object. Those objects are:

- View Repository – provides the *View_Rep*, *View_Rep_Manager* and *Main_Sighting_Processor* interfaces.
- View – provides the *View*, *View_Manager* and *View_Sighting_Processor* interfaces.
- Location – provides the *Location* and *Location_Sighting_Processor* interfaces.
- Locatable – provides the *Locatable* and *Locatable_Sighting_Processor* interfaces.

Moreover, user and management repository interfaces related to the given repository (user repository, equipment repository, etc.) defined in the *Domain_Repository* and *Domain_Repository_Managemen* modules are also implemented by the single object.

Figure 6 shows only one of many possible variants of distribution of objects among processes. Having CORBA interfaces already implemented it is relatively straightforward to implement another variant. The following numbers of real objects have been assumed in the figure:

- 50 locations
- 5 active views
- 100 sensors
- 100 badges
- 30 pieces of equipment
- 70 users

11 Configuration Manager

This programme enables full configuration of databases of Active Badge hardware as well as associated with them databases describing office environment. Its central part is a graphical editor of office map. This map is then used during addition, updating and localization of ABng hardware, equipment and users. It is also used for visualization of data gathered by ABng.

It standard view is presented using as an example window of Equipment Manager (Fig. 7).

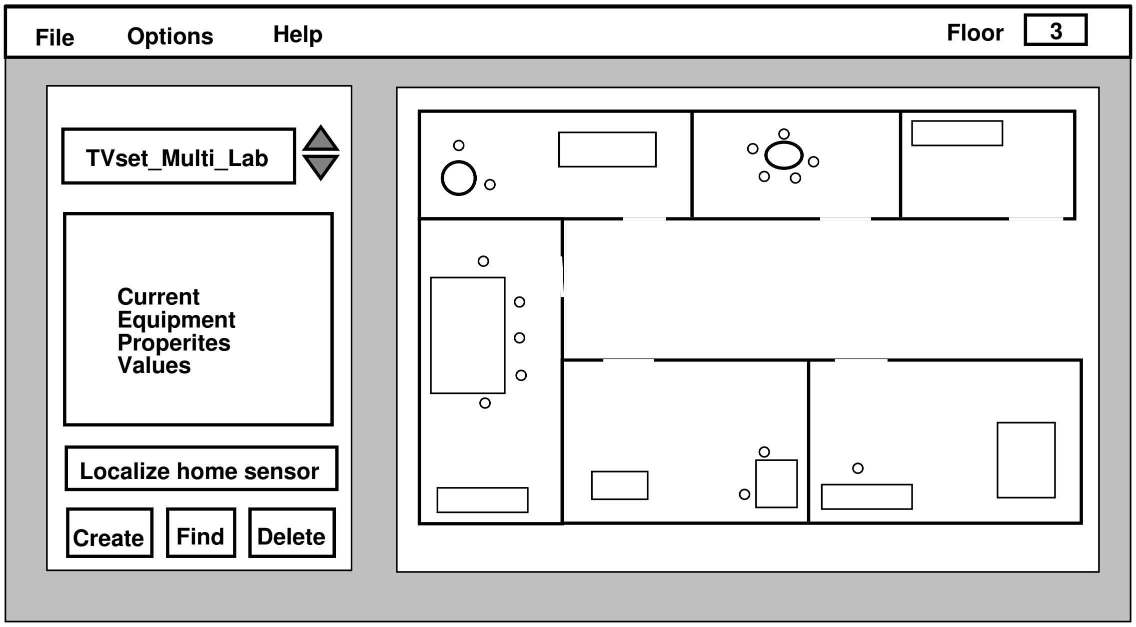

Figure 7: Equipment Manager window

References

[1] Frazer Bennett and Andy Harter. Low bandwidth infra-red networks and protocols for mobile communicating devices. Technical Report 93.5, Olivetti Research Laboratory, Cambridge, UK, 1993.