DELIVERABLE D1.1 (v1.0) Appendix B

IDL Design of Component Interfaces (Services)

Working Document

Status: 30 April 2002

Authors: *Wolfgang Ponweiser, Michael Zillich, Markus Vincze, Minu Ayromlou, Kingsley Sage, Jonathan Howell, Hilary Buxton, Gerald Umgeher, Christof Eberst, Antonis Argyros, Jan Palecek*

Project acronym: **ACTIPRET**
Project full title: **Interpreting and Understanding Activities of Expert Operators for Teaching and Education**

Action Line IV.2.1: **Real Time Distributed Systems (Cognitive Vision)**
Contract Number: **IST-2001-32184**

Contents

Appendix B: Definition of services

This appendix contains the definitions of services of components intended to be realised in the ActIPret Demonstrator (AD). These definitions are formulated in the Interface Definition Language (IDL) [www.omg.at].

The first subsection describes abstract patterns the service definitions are derived from. Subsection B.2 lists all created data types used in the services which are defined in subsection B.3. Subsection B.4 assigns the presented services to components proposed to the AD and subsection B.5 contains a communication sequence example of the AD of how all these services are intended to be used.

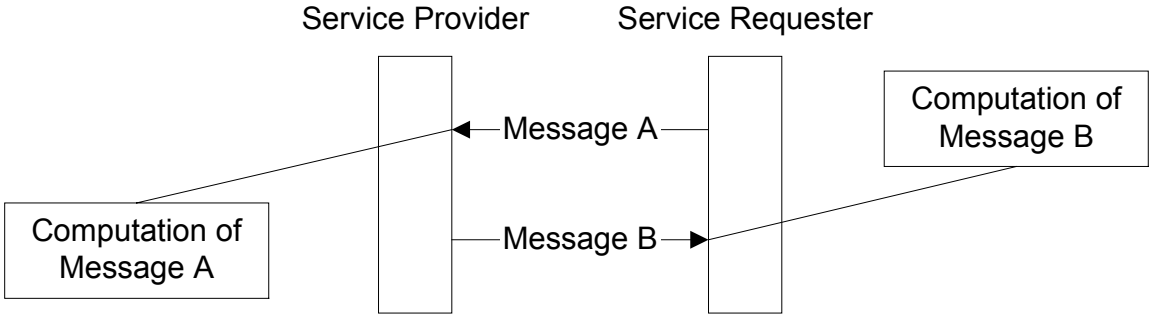
B.1 The Abstract Patterns

Service patterns are used to predefine a small subset of possible communication mechanisms. These patterns are templates used to form the service (interface between components). Hence the appropriate kind and number of patterns is important. The patterns introduced in this document are derived from the Smartsoft Communication Patterns [1]. They form the base of the communication planed by the OROCOS project [2]. Hence a big chance of reuse outside of the ActIPret project is expected.

The main principles taken into account during the development of these patterns are:

- *'The service principle'*.
- *'The hierarchy principle'*.

Every communication primitive can be subdivided into one or more messages. Every message has one receiver that provides the according message computation. Hence all messages sent from the Service Requester to the Service Provider are listed at the Service Provider. And all messages sent from the Service Provider to the Service Requester are listed at the Service Requester.

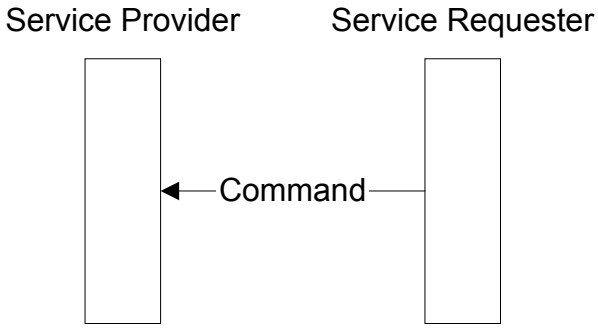


B.1.1 Datatypes

1. There exist a number of basic data types: *short, long, unsigned short, unsigned long, float, double, char, string, Boolean, octet*. A definition can be found in [3]. These data types can be used without any previous definition. But please consider that these data types are mapped to C++ defined by the IDL compiler (see [3]; e.g. *long* is translated to *CORBA::Long*).
2. All other data types must be defined separately using the *struct* key word. (Although that the key word *typedef* is defined in the IDL, it is not sufficiently supported by OSCAR, the suggested communication tool.)

B.1.2 Command pattern

- The command pattern consists of a single ***command*** from the Service Requester to the Service Provider.
- The command can, but does not have to, contain data.

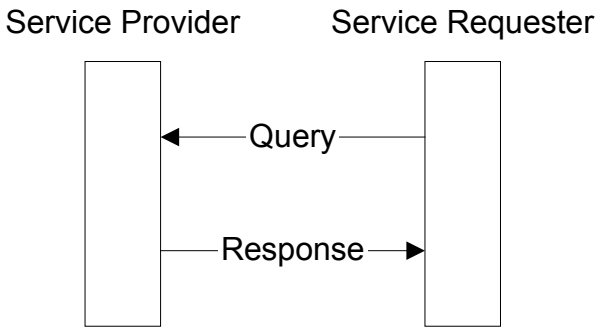


IDL:

```
interface ServiceName&Provider
{
      oneway void Command(in Type Data);
};
```

B.1.3 Query pattern

- The query pattern consists of the ***query*** message from the Service Requester to the Service Provider and the ***response*** sent from the Service Provider to the Service Requester.
- Both messages can, but do not have to, contain data.
- It is ***non-blocking***.



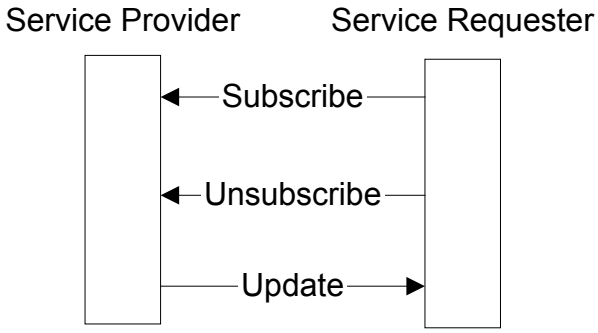
IDL:

```
interface ServiceName&Provider
{
      oneway void Query(in Type Data);
};

interface ServiceName&Requester
{
      oneway void Response(in Type Data);
};
```

B.1.4 Autoupdate pattern

- The autoupdate pattern consists of the ***subscribe*** and ***unsubscribe*** messages from the Service Requester to the Service Provider and the ***update*** sent from the Service Provider to the Service Requester.
- All messages can, but do not have to, contain data.
- It is ***non-blocking***.



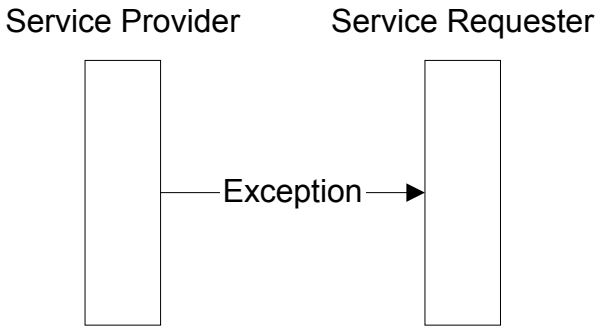
IDL:

```
interface ServiceName&Provider
{
      oneway void Subscribe(in Type Data);
      oneway void Unsubscribe(in Type Data);
};

interface ServiceName&Requester
{
      oneway void Update(in Type Data);
};
```

B.1.5 Exception pattern

- The pattern consists of a single ***exception*** message from the Service Provider to the Service Requester.
- The exception contains always two strings:
  1. The entire message it refers to
  2. An exception description
- Every service provides exactly one such message with the name: ‘Exception’.



IDL:

```
interface ServiceName
{
      oneway void Exception(in string Message, in string Description);
};
```

B.1.6 Coding Conventions

To simplify reading the messages we suggest specifying some naming-conventions. A unique keyword or a defined keyword substitution should express a specific communication pattern and the semantic of the message. The following table presents the suggested conventions where: XXX is a placeholder for the free parts of the message names; *Verb* is a placeholder for a verb. The **bold** written letters are the fixated parts of the message names.

Pattern	Message format	Semantic	Example
Command	*Verb*XXX	Do verb	*Find*Object
	Stop*Verb*XXX	Stop doing verb	**Stop***Find*Object
	SetXXX	Send data	**Set**Pose
Query	*Verb*XXX	Do verb and send one response	*Calculate*Pose
	GetXXX	Return data	**Get**Pose
Query-Response	*Verb*XXX**Response**	Response of a verb-query	*Calculate*Pose**Response**
	GetXXX**Response**	Response of a get-query containing the requested data	**Get**Pose**Response**
Autoupdate	*Verb*XXX	Do verb and send multiple response	*Avoid*Collision
	AutoUpdateXXX	Return data repeated	**AutoUpdate**Image
	Cancel*Verb*XXX	Stop doing verb and cancel sending of responses	**Cancel***Avoid*Collision
	CancelAutoUpdateXXX	Cancel sending of responses	**CancelAutoUpdate**Image
Autoupdate-Response	*Verb*XXX**Response**	Response of a verb-autoupdate	*Avoid*Collision**Response**
	AutoUpdateXXX**Response**	Response of an update-autoupdate	**AutoUpdate**Image**Response**

		containing the requested data	

B.1.6.1 Datatype and Service Names

To simplify discrimination between datatype and service definitions only datatype names should end with the term **Type** (e.g. Pose**Type** but ImageServer).

B.1.6.2 Message (function) overloading

Function overloading is not permitted in IDL. Specify two different messages with the same name and different parameters like in C++ is not allowed. Hence every message of a service has to have a service wide unique name!

B.1.6.3 Linking Responses to their Questions

Please consider, that Query and Autoupdate are both non-blocking patterns. So the method of the service requester that receives the response needs a hint to be able to assign the received response to the original request (query or autoupdate). Hence e.g. if you send a query 'GetObjectPose(ObjectID)' than the response should look like 'GetObjectResponse(ObjectID, Pose)'. Here the response contains the ObjectID to enable assignment of the response.

B.2 User Defined Datatypes

The first definitions of this document define all composite data types in alphabetical order.

B.2.1 CameraParameterType

```
struct CameraParameterType
{
      double ImagePixelSize;
      double SU;                // often called sx
      double SV;                // spezifies the actual compression
                                // of the pixels (no squares), or
                                // chip rotated to the optical axis
      double HP_U;              // also called HP_X: where the beam
                                // through the center hits the chip
      double HP_V;              // also called HP_Y
      double RadialDistortion; // kappa
      double FocalLength;

      IDType CameraID;
};

typedef sequence<CameraParameterType> CameraParameterSeqType;
```

B.2.2 CameraViewPointType*[1]

```
struct CameraViewPointType
{
      IDType      CameraID;

      // The Pose orientation is directed from the camera to the object
      Pose3DType  Pose;
      Pose3DType  Accuracy;
```

[1] A * after data type or service names indicate that the according definition is still under development.

```
      double      Distance;
      double      DistanceAccuracy;
};

typedef sequence<CameraViewPointType> CameraViewPointSeqType;
```

B.2.3 ConceptFunctionType

```
struct ConceptFunctionType
{
      string            FunctionName;
      sequence<string>  Arguments; // One or more arguments
};

typedef sequence<GestureHandUpdateType>  GestureHandUpdateSeqType;
```

B.2.4 GestureProbabilitySeqType

```
struct GestureProbabilityType
{
      ObjectIDType            ObjectID;   // Label from reasoning engine
      IDType                  MethodeModelID;
      sequence<double>        ProbabilitySeq;
      unsigned short          MaxGestureType;         // Gesture type with
                                                      // highest prob
      double                  GestureConfidence;      // In range [0,1]
};

typedef sequence<GestureProbabilityType> GestureProbabilitySeqType;
```

B.2.5 IDType

```
struct IDType
{
      unsigned long     ID;
};

typedef sequence <IDType> IDSeqType;
```

B.2.6 ImagePatternType*

```
enum ImagePatternType{IRGB,IGREY,IYUV422,IYUV411,IYUV444,IBAYERPATTERN};

// IRGB: Red, Green, Blue
// IGREY: greyscale
// IYUV422: y1, u, y2, v
// BAYER: ???
```

B.2.7 ImagePointType

```
struct ImagePoint2DType
{
      long X;
      long Y;
};
typedef sequence<ImagePoint2DType> ImagePoint2DSeqType;
```

B.2.8 ImageRawType

```
struct ImageRawType
{
      ImagePatternType  ImagePattern;
      sequence<octet>   ImageData;
      ROIType           ImageDetail;
};
```

```
typedef sequence<ImageRawType> ImageRawSeqType;
```

B.2.9 ImageSharedMemoryType

```
struct ImageSharedMemoryType
{
      ImagePatternType  ImagePattern;
      SharedMemoryType  ImageSharedMemory;
      ImageSizeType     ImageSize;
};

typedef sequence<ImageSharedMemoryType> ImageSharedMemorySeqType;
```

B.2.10 ImageSizeType

```
struct ImageSizeType
{
      long ImageHeight;
      long ImageWidth;
};

typedef sequence<ImageSizeType> ImageSizeSeqType;
```

B.2.11 ModelIDType

```
struct ModelIDType
{
      string Name;  // e.g. "CD", "Hand", "CD-Player"
};

typedef sequence <ModelIDType> ModelIDSeqType;
```

B.2.12 ModelHypothesisSeqType

```
// just a list of numbered modelID's (model-hypotheses)
// model-hypothesis: hypothesis of what the object is (what is the model)
struct ModelHypothesisType
{
      IDType            HypothesisID;
      ModelIDType       ModelID;
      double            ModelConfidence;
}

typedef sequence<ModelHypothesisType>    ModelHypothesisSeqType;
```

B.2.13 ModelType*

```
enum DataModelType {WIREFRRAME, COLOUR};  // and many more later on

struct ModelType
{
      DataModelType     type;
      string            ModelData;  // for now this is a just string
                                    // as the most generic datatype,
                                    // a more detailed definition      will
                                    // emerge later
};
```

B.2.14 NearAttributesType

```
// can be redefined after year 1 of ActIPret
struct NearAttributesType
{
      unsigned float    Threshold;
};
```

B.2.15 ObjectPoseHypothesisSeqType

```
// multiple tripple: unique ObjectID, unique ModelID, multiple
PoseHypotheses
// typical output of the question: where is this
struct ObjectPoseHypothesisType
{
      ObjectIDType           ObjectID;
      ModelIDType            ModelID;
      PoseHypothesisSeqType  PoseHypothesesList;
}

typedef sequence<ObjectPoseHypothesisType> ObjectPoseHypothesisSeqType;
```

B.2.16 ObjectModelHypothesisType

```
// multiple tripple: unique ObjectID, multiple ModelHypotheses, unique Pose
// typical output of the question: what is there (if there is pose)
struct ObjectModelHypothesisType
{
      ObjectIDType           ObjectID;
      ModelHypothesisSeqType ModelHypothesesList;
      ObjectPoseType         Pose;
}

typedef sequence<ObjectModelHypothesisType> ObjectModelHypothesisSeqType;
```

B.2.17 ObjectModelHypothesis2DType

```
// multiple tripple: unique ObjectID, multiple ModelHypotheses, unique Pose
// typical output of the question: what is there (if there is pose)
struct ObjectModelHypothesis2DType
{
      ObjectIDType           ObjectID;
      ModelHypothesisSeqType ModelHypothesesList;
      ROIType                Pose;
}

typedef sequence< ObjectModelHypothesis2DType >
ObjectModelHypothesis2DSeqType;
```

B.2.18 ObjectIDType

```
struct ObjectIDType
{
      IDType      ID;
      Boolean     FromRequester;    // There is no 'global' ObjectID
                                    // Hence this flag indicates if
                                    // the ID is originated by the
                                    // ServiceRequester
                                    // true: ServiceRequester-ObjectID
                                    // false: ServiceProvider-ObjectID
};
```

B.2.19 ObjectLinkType

```
struct ObjectLinkType
{
      LinkType          Component;  // to be defined
                                    // contains a reference to
                                    // a component or a reference
                                   // to a service list entry
      ObjectIDType      ObjectID;   // in this case the ID is the
                                    // one from the component the
                                    // link refers to
                                    // the FromRequester -bit
                                    // is always false
      ModelIDType       ModelID;
```

```
}

typedef sequence<ObjectLinkType> ObjectLinkSeqType;
```

B.2.20 ObjectPairType

```
// Pose of RelatedObject referred to the ReferenceObject
struct ObjectPairType
{
      ObjectIDType      RefObjectID;
      ObjectIDType      RelatedObjectID;
      ObjectPoseType    RelatedObjectPose;
};

typedef sequence<ObjectPairType> ObjectPairSeqType;
```

B.2.21 ObjectPoseType

```
struct ObjectPoseType
{
      PoseType          Pose;
      PoseType          Accuracy;
      double            PoseConfidence;
};
```

B.2.22 ObjectToTrajectoryRelationType

```
struct ObjectToTrajectoryRelationType
{
      ObjectIDType            ObjectID;
      ModelHypothesisSeqType  ModelHypothesesList;
      ObjectPoseType          Pose;
      double                  Distance;   // shortest distance
                                          // to trajectory
};
```

B.2.23 ObjectType

```
// multiple tripples of: unique ObjectID, unique ModelID, unique Pose
// typical to transmit multiple objects
struct ObjectType
{
      ObjectIDType      ObjectID;
      ModelIDType       ModelID;
      ObjectPoseType    Pose;
}

typedef sequence<ObjectType> ObjectSeqType;
```

B.2.24 Point2DType

```
struct Point2DType
{
      double x;
      double y;
};
```

B.2.25 Point3DType

```
struct Point3DType
{
      double X;
      double Y;
      double Z;
};

typedef sequence<Point3DType> Point3DSeqType;
```

B.2.26 Pose3DType

```
struct Pose3DType
{
      Point3DType       Point;
      Rotation3DType    Rotation;
};

typedef sequence<Pose3DType> Pose3DSeqType;
```

B.2.27 PoseHypothesisSeqType

```
// just a list of numbered poses (pose-hypothesis)
struct PoseHypothesisType
{
      IDType            HypothesisID;
      ObjectPoseType    Pose;
      double            Confidence;
};

typedef sequence<PoseHypothesisType>     PoseHypothesisSeqType;
```

B.2.28 ROIType

```
struct ROIType
{
      ImagePointType    Offset;
      ImageSizeType     ImageSize;

      // not used in the first version of the Image Server
      double            Angle;
      unsigned short    SamplingHeight;
      unsigned short    SamplingWidth;
};

typedef sequence<ROIType> ROISeqType;
```

B.2.29 Rotation3DType

```
struct Rotation3DType
{
      double Alpha;     // represents Azimuth
      double Beta;      // represents negative Elevation
      double Theta;     // represents the Roll angle
};

typedef sequence<Rotation3DType> Rotation3DSeqType;
```

B.2.30 ServicePropertyType

```
struct ServicePropertyType
{
      string            ServiceName;
      double            ProcCosts;
      double            QoS;
      unsigned short    Priority;
};

typedef sequence<ServicePropertyType> ServicePropertySeqType;
```

B.2.31 SharedMemoryType

```
struct SharedMemoryType
{
      long SharedMemoryKey;
      long SharedMemorySize;
};
```

```
typedef sequence<SharedMemoryType> SharedMemorySeqType;
```

B.2.32 SOIType

```
// Space Of Interrest:
// for now: rectangular axis - parallel box
struct SOIType
{
      PoseType          Origin;
      Double            LengthX;
      Double            LengthY;
      Double            LengthZ;
};
```

B.2.33 TimestampType

```
// TimestampType is part of the OSCAR interface
/*
 * time in msec after 3:00 UTC (=GMT)
 * overflow avoidance garanteed 'cause
 * long is mapped to at least 4Byte
 * take care while running at 4:00 (winter)
 * and 5:00 (summer)! wrap around occurs then!
 */
typedef unsigned long TimestampType;

/*
 * TimestampSeq are transmitted as sequences.
 * normally, the sequence contains
 * exactly one element. Integration
 * modules  may containt several
 * timestamps to be defined, perhaps
 * a source tag may be added at this place.
 */
typedef sequence<TimestampType> TimestampSeq;
```

B.2.34 TrajectoryType

```
struct TrajectoryPointType
{
      TimestampType     Time;
      Pose3DType        Pose;
}

typedef sequence<TrajectoryPointType> TrajectoryType;
```

B.2.35 TrinocularHeadGeometryType

```
struct TrinocularHeadGeometryType
{
      // Transformation from the Camera Head Base to the Verge Axis
      Pose3DType HeadBaseToVergeAxisLeft;
      Pose3DType HeadBaseToVergeAxisRight;
      Pose3DType HeadBaseToVergeAxisTop;

      // Transformation from the Verge Axis to the focal point
      Pose3DType VergeAxisToCameraLeft;
      Pose3DType VergeAxisToCameraRight;
      Pose3DType VergeAxisToCameraTop;

      // ID of the Camera
      IDType      LeftID;
      IDType      RightID;
      IDType      TopID;
};

typedef sequence< TrinocularHeadGeometryType >
      TrinocularHeadGeometrySeqType;
```

B.2.36 TrinocularHeadCameraPoseType

```
struct TrinocularHeadCameraPoseType
{
      Pose3DType  HeadPose;
      double      VergeLeft;
      IDType      LeftID;
      double      VergeRight;
      IDType      RightID;
      double      VergeTop;
      IDType      TopID;
};

typedef sequence<CameraHeadPoseType> CameraHeadPoseSeqType;
```

B.3 Services

The following definitions are all services of all components intended to be used within the AD in alphabetical order.

B.3.1 Service: CameraPoseServer

```
interface CameraPoseProvider
{
      // query
      oneway void GetCameraPose(in IDType CameraID);
      oneway void GetCameraParameters(in IDType CameraID);
      oneway void GetCameraTrinocularHeadPose();
      oneway void GetTrinocularHeadGeometry();

      // autoupdate
      oneway void AutoUpdateCameraPose(in IDType CameraID);
      oneway void AutoUpdateCameraTrinocularHeadPose();
      // stop autoupdate
      oneway void CancelAutoUpdateCameraPose(in IDType CameraID);
      oneway void Cancel AutoUpdateCameraTrinocularHeadPose();
};

interface CameraPoseRequester
{
      // query answer
      oneway void GetCameraPoseResponse(in IDType CameraID,
                                        in Pose3DType CameraPose);
      oneway void GetCameraParametersResponse(
                        in CameraParameterType CameraParameters);
      oneway void GetCameraTrinocularHeadPoseResponse(
                        in TrinocularHeadCameraPoseType HeadPose);
      oneway void GetTrinocularHeadGeometryResponse(
                        in TrinocularHeadGeometryType HeadGeometry);

      // autoupdate answer
      oneway void AutoUpdateCameraPoseResponse(
                        in CameraPoseType CameraPose);
      oneway void AutoUpdateCameraTrinocularHeadPoseResponse(
                        in TrinocularHeadCameraPoseType HeadPose);

      // exception
      oneway void Exception(in string Message,
                            in string Description);
                            // standard exception
};
```

B.3.2 Service: CaptureControl

```
// for the system Phase
#define SP_STARTUP           0
#define SP_LEARNING_PHASE    1
```

```
#define SP_EXPERT_PHASE       2
#define SP_TUTOR_PHASE        3
#define SP_SHUTDOWN           4

// for exceptions
#define CS_WRONG_PHASE        1     // Capture can't take place in
                                     // this phase
#define CS_NO_STATIC_DATABASE 2     // Tutor mode before expert mode !

interface CaptureControlProvider
{
      // Command pattern
      oneway void SetSystemPhase(in unsigned short SystemPhase);

      // Query pattern
      oneway void CaptureExemplar();     // Phase is already known
};

interface CaptureControlRequester
{
      // Response to query pattern
      oneway void CaptureExemplarResponse();

      // exception
      oneway void Exception(  in string Message,
                              in string Description);
                              // standard exception
};
```

B.3.3 Service: DetectObject

```
interface DetectObjectProvider
{
      // query
      oneway void DetectObject(in IDType ObjectID,
                              in IDType ModelID,
                              in SOIType SOI)
                              // configuring & control (start), one-shot
                              // just detect object (where is this?)
};

interface DetectObjectRequester
{
      // query answer
      oneway void DetectObjectResponse(in ObjectPoseHypothesisType
                                          ObjectPoseHypotheses,
                                       in TimestampType Time);
                                       // data push
      // exception
      oneway void Exception(in string Message,
                            in string Description);
                            // standard exception
};
```

B.3.4 Service: DetectProximity*

```
interface DetectProximityProvider
{
      // command
      oneway void StopAutoUpdateObject( in ObjectModelHypothesisType
                                           Object);
                                       // control(stop)
                                       // stop computation on this
                                       // object found by the
                                       //AutoUpdateObjectsNearTrajectory
                                       // or
                                       // AutoUpdateNearObjectPairs
                                       // message
      // query
```

```
        oneway void GetDistance(in ObjectType RefObjectID,
                                in ObjectType RelatedObjectID);
                                // just calculate distance,
                                // one shot
        oneway void GetNearObjectPairs(
                                in ObjectSeqType ObjectList;
                                in NearAttributesType Attributes);
                                // one shot
                                // checks objects pairwise
                                // for NearAttributes
        // autoupdate
        oneway void AutoUpdateNearObjectPairs(
                                in ObjectLinkSeqType ObjectList;
                                in NearAttributesType Attributes);
                                // continuous
                                // the object poses can be requested from
                                // the linked component;
                                // checks objects pairwise
                                // for NearAttributes
        oneway void CancelAutoUpdateNearObjectPairs(
                                in ObjectLinkSeqType ObjectList);
        oneway void AutoUpdateObjectsNearTrajectory(
                                in ObjectLinkType RefObjectLink,
                                in TrajectoryType RefObjectTrajectory,
                                in NearAttributesType Attributes);
                                // configuring & control (start),
                                // continuous
                                // get objects that fulfill NearAttributes
                                // to the trajectory
                                // the trajectory can degrade to
                                // one object pose
        oneway void CancelAutoUpdateObjectsNearTrajectory(
                                in ObjectLinkType RefObjectLink);
};

interface DetectProximityRequester
{
        // query answer
        oneway void GetDistanceResponse(
                                in ObjectIDType RefObjectID,
                                in ObjectIDType RelatedObjectID,
                                in ObjectPoseType DistanceVector);
                                // data push
        oneway void GetNearObjectPairsResponse(
                                in ObjectSeqType ObjectList;
                                in NearAttributesType Attributes,
                                in ObjectPairSeqType ObjectPairList,
                                in TimestampType Time);
                                // data push
        // autoupdate answer
        oneway void AutoUpdateNearObjectPairsResponse(
                                in ObjectLinkSeqType ObjectList,
                                in NearAttributesType Attributes,
                                in ObjectPairSeqType ObjectPairList,
                                in TimestampType Time);
                                // data push
        oneway void AutoUpdateObjectsNearTrajectoryResponse(
                                in ObjectLinkType RefObjectLink,
                                in TrajectoryType RefObjectTrajectory,
                                in NearAttributesType Attributes,
                                in ObjectToTrajectoryRelationType
                                            ObjectList,
                                in TimestampType Time);
                                // data push
        // exception
        oneway void Exception(in string Message,
                              in string Description);
                              // standard exception
};
```

B.3.5 Service: FindObjectsOnTrajectory*

```
struct ProximityDistanceType
{
      double      Distance;   // Some kind of non-linear Euclidean
                              // metric
};

struct MutualProximityType
{
      IDType                  ObjectID1;
      IDType                  ObjectID2;
      ProximityDistanceType   Proximity;  // Non-linear distance
};

sequence <MutualProximityType> MutualProximitySeqType;

struct MutualProximityRelationType
{
      TimestampType           Timestamp;        // Timestamp from ORG
      MutualProximitySeqType  MutualProximityList;
};

interface FindObjectsOnTrajectoryProvider
{
      // subscribe
      oneway void FindObjectsOnTrajectory(in IDType ObjectID,
                                          in VIMarkerID,
                                          in double Threshold);
                                          // Threshold was added as
                                          // per INFA suggestion.
      // un-subscribe
      oneway void CancelFindObjectsOnTrajectory(in IDType ObjectID,
                                                in VIMarkerID);

      // subscribe
      oneway void FindObjectsByMutualProximity();
      // un-subscribe
      oneway void CancelFindObjectsByMutualProximity();
};

interface FindObjectsOnTrajectoryRequester
{
      oneway void FindObjectsOnTrajectoryResponse(
                        in TrajectoryRelationType Data);
      oneway void FindObjectsByMutualProximityResponse(
                        in MutualProximityRelationType Data);

      // exception
      oneway void Exception(in string Message,
                            in string Description);
                            // standard exception
};
```

B.3.6 Service: ImageServer

```
interface ImageProvider
{
      // query
      oneway void GetShmImage(in IDType CameraID);
      oneway void GetRawImage(in IDType CameraID,
                              in ROIType  ROI);

      // autoupdate
      oneway void AutoUpdateShmImage(in IDType CameraID);
      // stop autoupdate
      oneway void CancelAutoUpdateShmImage(in IDType CameraID);
```

```
};

interface ImageRequester
{
      // query answer
      oneway void GetShmImageResponse(in IDType CameraID,
                                      in ImageSharedMemoryType ShmImage);
      oneway void GetRawImageResponse(in IDType CameraID,
                                      in ImageRawType RawImage);

      // autoupdate answer
      oneway void AutoUpdateShmImageResponse(in IDType CameraID,
                                             in ImageSharedMemoryType
                                                ShmImage);

      // exception
      oneway void Exception(in string Message,
                            in string Description);
                            // standard exception
};
```

B.3.7 Service: MaintainPlan

```
// for the exceptions
#define MP_SUCCESS           1     // Maintenance query executed OK
#define MP_NO_SUCH_EXEMPLAR  2     // Invalid exemplar ID

interface MaintainPlanProvider
{
      // Command
      oneway void DisplayWholePlan();
      oneway void DisplayPlanExemplar(in unsigned short ExemplarID);
      oneway void DeleteWholePlan();
      oneway void DeletePlanExemplar(in unsigned short ExemplarID);
};

interface MaintainPlanRequester
{
      // Exception
      oneway void Exception(  in string Message,
                              in string Description);
                              // standard exception
};
```

B.3.8 Service: ModelServer*

```
interface ModelProvider
{
      // command
      oneway void RegisterModel(in ModelIDType ModelID,
                                in DataModelType Type);
      // query
      oneway void GetModelTypes(in ModelIDType ModelID);
      oneway void GetModel(in ModelIDType ModelID,
                           in DataModelType Type);
};

interface ModelRequester
{
      // query
      oneway void GetModelTypesResponse(in ModelIDType ModelID,
                        in sequence<DataModelType> ModelTypeList);
      oneway void GetModelResponse(in ModelIDType ModelID,
                        in DataModelType Type,
                        in ModelType Model);

      // exception
      oneway void Exception(in string Message,
                            in string Description);
```

```
                              // standard exception
};
```

B.3.9 Service: ProduceConceptFunctions

```
// SpecifyModelParametersHdr
#define           // to be defined

interface ProduceConceptFunctionsProvider
{
      // Command pattern
      oneway void SetNullHypotheses();

      // Auto-update pattern
      oneway void ProduceConceptFunctions();          // To subscribe
      oneway void CancelProduceConceptFunctions();    // To unsubscribe

      // Some functions for providing model parameters ?
};

interface ProduceConceptFunctionsRequester
{
      // Auto-update response
      oneway void ProduceConceptFunctionResponse(
                              in ConceptFunctionType Data);

      // Some functions for requesting model parameters … to be defined

      // exception
      oneway void Exception(  in string Message,
                              in string Description);
                              // standard exception
};
```

B.3.10 Service: Recognition2D*

```
interface RecognitionProvider
{
      // command
      oneway void SetTimeLimit(in unsigned long TimeLimit);

      //query
      oneway void GetTimeLimit();
      oneway void GetAllObjectModels();
      oneway void GetNumberOfObjectModels();
      oneway void GetAllCategoryIDs();
      oneway void GetNumberOfCategoryIDs();
      oneway void RecognizeFromCamera(in ROIType   ROI,
                                      in IDSeqType IDs);
      oneway void RecognizeFromImage(in ImageRawType RawImage,
                                     in IDSeqType    IDs);
      oneway void RecognizeFromShMen(in ImageSharedMemoryType ShmImage,
                                     in ROIType               ROI,
                                     in IDSeqType             IDs);
};

interface RecognitionRequester
{
      // answer
      oneway void GetTimeLimitResponse(in unsigned long TimeLimit);
      oneway void GetAllObjectModelsResponse(in IDSeqType IDs);
      oneway void GetNumberOfObjectModelsResponse(in unsigned short Num);
      oneway void GetAllCategoryIDsResponse(in IDSeqType IDs);
      oneway void GetNumberOfCategoryIDsResponse(in unsigned short Num);
      oneway void RecogniseFromCameraResponse(
                        in ObjectWithPosition2DSeqType ObjectPositionList);
      oneway void RecogniseFromImageResponse(
                        in ObjectWithPosition2DSeqType ObjectPositionList);
      oneway void RecogniseFromShMenResponse(
```

```
                        in ObjectWithPosition2DSeqType ObjectPositionList);

      // exception
      oneway void Exception(  in string Message,
                              in string Description);
                              // standard exception

};
```

B.3.11 Service: Recognition3D*

```
interface Recognition3DProvider
{
      //query
      oneway void Recognise(  in SOIType SOI,
                              in IDSeqType IDs);
};

interface Recognition3DRequester
{
      // answer
      oneway void RecogniseResponse(
                        in ObjectWithPosition3DSeqType ObjectPoseList);
};
```

B.3.12 Service: TrackObject

```
interface TrackObjectProvider
{
      // command
      oneway void StopReportingOnHypothesis(in IDType ObjectID,
                                            in IDType HypothesisID);
                                            // control (stop)
      // autoupdate
      oneway void TrackObject(in IDType ObjectID,
                              in IDType ModelID,
                              in SOIType SOI,
                              in unsigned short Frequency);
                              // configuring & control (start),
                              // continuous
                              // D E T E C T   &   T R A C K
                              // object and calculate poses
      oneway void CancelTrackObject(in IDType ObjectID);
                             // cancel autoupdate
                             // control (stop)
      oneway void TrackObjectInitialised(in IDType ObjectID,
                              in IDType ModelID,
                              in PoseType InitialPose,
                              in unsigned short Frequency);
                              // configuring & control (start),
                              // continuous
                              // just track object and calculate poses
      oneway void CancelTrackObjectInitialised(in IDType ObjectID);
                              // cancel autoupdate
                              // control (stop)
};

interface TrackObjectRequester
{
      // autoupdate answer
      oneway void TrackObjectResponse(in ObjectPoseHypothesisType
                                          ObjectHypothesisList,
                                      in TimestampType Time);
                                          // data push
      // exception
      oneway void Exception(in string Message,
                            in string Description);
                            // standard exception
};
```

B.3.13 Service: UpdateGesturesForHandObject*

```
interface UpdateGesturesForHandObjectProvider
{
      // command
      oneway void SetTorsoPosition(in Point3DType TorsoPosition,
                                   in Point3DType TorsoPositionAccuracy,
                                   in TimestampType Time);
                                   // in future the torso will be delivered
                                   // by another detction & tracking
                                   // service
      // autoupdate
      oneway void AutoUpdateGesturesForHandObject(
                        in ObjectSeqType Data,
                        in TimestampType Time);
      oneway void CanelAutoUpdateGesturesForHandObject(
                        in ObjectSeqType Data);
};

interface UpdateGesturesForHandObjectRequester
{
      // autoupdate response
      oneway void AutoUpdateGesturesForHandObjectResponse(
                        in GestureProbabilitySeqType Data,
                        in TimestampType Time);

      // exception
      oneway void Exception(  in string Message,
                              in string Description);
                              // standard exception
};
```

B.3.14Service: RecogniseGesture*
(former UpdateGesturesForHandObject)

```
interface RecogniseGestureProvider
{
      // command
      oneway void SetTorsoPosition(in Point3DType TorsoPosition,
                                   in Point3DType TorsoPositionAccuracy,
                                   in TimestampType Time);
                                   // in future the torso will be delivered
                                   // by another detction & tracking
                                   // service
      // autoupdate
      oneway void AutoUpdateGesturesForHandObject(
                        in ObjectLinkType ObjectReference);
      oneway void CancelAutoUpdateGesturesForHandObject(
                        in ObjectLinkType ObjectReference);
};

interface RecogniseGestureRequester
{
      // autoupdate response
      oneway void AutoUpdateGesturesForHandObjectResponse(
                        in GestureProbabilitySeqType Data);

      // exception
      oneway void Exception(  in string Message,
                              in string Description);
                              // standard exception
};
```

B.3.15 Service: ViewController

```
interface ViewControlProvider
```

```
{
      // command
      oneway void RequestViewPoint( in CameraViewPointType ViewPoint,
                                    in ServicePropertyType Property);
};

interface ViewControlRequester
{
      // exception
      oneway void Exception(in string Message,
                            in string Description);
                            // standard exception
};
```

B.4 Component Services

B.4.1 User HMI

The User HMI does not provide any isolated services (at least for now), it either only requests them (command patterns) or reacts to responses (as the response part of a query pattern). This may change later if we try to centralise User HMI functions onto one single interface. For now, interfaces are likely to be distributed throughout the system.

B.4.2 Activity Planner

B.4.2.1 Service: CaptureControl

(See B.3.1) When the User HMI starts the expert phase in the activity planner, capture of exemplars will take place to build the static database. When the User HMI starts the tutor phase (assuming that the static database has already been built), the functions of stepwise (or other) comparison of the static database with the current temporary working plan is an internal function of the activity planner.

In the expert phase this service generates data for the static activity plan (exemplar data). In the tutor phase this service generates data for the temporary workspace plan that is compared to the static plan either incrementally or completely.

B.4.2.2 Service: Maintain Plan

(See B.3.7) Maintenance only applies to linear example in the static database and can only occur in the expert phase. This pattern assumes that the code for displaying plans resides within the activity planner, and that the plan is not returned as a data object to the User HMI as a data object. A future refinement would seek to centralise all aspects of the user interface into the User HMI component.

We have not yet considered options for editing of the activity plan. We intend to add later new messages like ‘IntegrateLinearExemplars’ that may build some hierarchical structures from the existing linear set.

B.4.3 Activity Reasoning Engine

B.4.3.1 Service: Produce Concept Functions

(See B.3.9) This service proposes a struct to hold a plan concept function. This struct may evolve with time but only impinges on COGS components. Concept functions can only be produced in the expert and tutor phases, but the reasoning engine is not explicitly phase aware.

Message: Set NULL hypotheses

This command pattern causes the set of plan hypothesis functions to be deleted (also causing all entries in the reasoning engine visual index to be deleted). This would normally happen just before start of a “capture a scenario exemplar” pattern, but it may prove useful for diagnostic and development purposes.

Message: Specify behavioural model parameters

A “complete” activity plan needs to include any specific behavioural models derived from generic ones during the learning phase. The underlying assumption here is that all model parameters can be represented as a <sequence> of binary values (name, numeric parameter). We need to develop further our ideas on how these models are stored.

B.4.4 Gesture Recogniser

B.4.4.1 Service: Recognise Gesture

(See B.3.13,B.3.14) Determine set of probabilities for on-going hand gestures.

B.4.5 Object Relation Generator

B.4.5.1 Service: Detect Proximity (or Find Objects on Trajectory)

(See B.3.4) Finds objects along the trajectory path of a specified hand and finds objects in mutual proximity to one another.

B.4.6 Recogniser

B.4.6.1 Service: Recognition3D

(See B.3.11) Finds objects already learned in the learning phase inside of a defined space of interest (3D). (Answers the question: Where is what within a restricted zone?)

B.4.6.2 Service: Recognition2D (or Recognition)

(See B.3.10) Finds objects already learned in the learning phase inside of a defined region of interest (2D).

B.4.6.3 Service: Recognition Learning

(See B.3.11) Generates a model of objects necessary for later recognition. It is intended to use it in the learning phase.

B.4.7 Hand Detector and Tracker

B.4.7.1 Service: Track Object (or Hand Detector)

(See B.3.12) Finds (detects) and tracks hand objects. (Answers the question: Where are hands within a restricted zone?)

B.4.8 Object Detector and Tracker

B.4.8.1 Service: Track Object

(See B.3.12) Finds (detects) and tracks objects if a wireframe model is available. (Answers the question: Where are such objects within a restricted zone?)

B.4.8.2 Service: Detect Object

(See B.3.3) Just finds (detects) objects if a wireframe model is available. (Answers the question: Where are such objects within a restricted zone?)

B.4.9 Model Server

B.4.9.1 Service: Model Server

(See B.3.8) The model server is intended to store object model data. There are various types of object models (wireframe, colour, PCA, distinguished regions...). Some model types are rather generic and not method specific (wireframe, colour - many recognition and tracking methods can use these models). Others are very specific for a certain method (PCA, distinguished regions - method and model fit together, nobody else could use that model). Only generic model types are stored in the model server, specific model types are stored in the according detector/recogniser/tracker component. In the latter case the model server just has a link, indicating that there exists in the system a PCA model. So the model server knows for every model ID what model types are available. The generic ones it can provide itself, for the specific ones look in the service list that offers these kinds of model types.

Message: Get model types

Ask the model server to return a list of model types for a given model ID. E.g. "what types of model do we have for a Hand?" - "There is a colour model, a PCA model and a wireframe model".

Meassge: Get model

Get actual model data.

Meassge: Register Model

Tell the model server that there exists a model of given type for the given model id. The actual model data resides in the requesting component (e.g. A PCA based detector).

B.4.10 Pose Server

B.4.10.1 Service: Pose Server

(See B.3.1) Delivers all poses related to the cameras and camera head it is assigned to.

B.4.11 Image Server

B.4.11.1 Service: Image Server

(See B.3.6) Delivers the entire image in a shared memory or directly a predefined part of the image of the camera the image server is assigned to.

B.4.12 ServiceList with ViewController

B.4.12.1 Service: ViewController

(See B.3.15) Realises requested camera poses according to hardware constraints.

B.5 A communication sequence example of the AD

The following communication sequence is a suggested sequence of service requests and messages of a typical start-up of the AD in the expert phase where the AD user inserts a CD in a CD-player. The sequence should provide a better understanding of the framework and architectural concepts of this deliverable 1.1.

Please keep in mind that:
1) The presented sequence is just an example. No specific sequential dependencies should be derived.

2) All messages are unsynchronised. Hence the sequential order is intuitive but can be totally different when executed (partly parallel).

Used Component abbreviations:

HMI	Human Machine Interface
AP	Activity Planner
ARE	Activity Reasoning Engine
GR	Gesture Recogniser
ORG	Object Relation Generator
REC	Recogniser
HT	Hand Detector and Tracker
OT	Object Detector and Tracker
MS	ModelServer
IS	ImageServer(Nr. 1,2,3,…)
PS	PoseServer

Legend:

X(Y) ser	X requests the service ‘ser’ at Y
X(Y) cancel ser	X cancels the service ‘ser’ at Y
X -> Y (msg)	X sends the message ‘msg’ to Y

1	HMI(AP)	CaptureControl
2	HMI -> AP	SetSystemPhase(SP_EXPERT_PHASE)
3	AP(ARE)	ProduceConceptFunctions
4	AP -> ARE	ProduceConceptFunctions()
5	ARE(MS)	ModelServer
6	ARE -> MS	GetModelTypes(Hand)
7	MS -> ARE	GetModelTypesResponse(Colour, Motion)

```
8     ARE(HT)        TrackObject
9     ARE -> HT      TrackObject(1, Hand, Workspace)
10    HT(IS1)        ImageServer
11    HT -> IS1      GetShmImage()
12    IS1 -> HT      GetShmImageResponse(ShmKey)
13    HT(PS)         CameraPoseServer
14    HT -> PS       GetCameraPose(1)
15    PS -> HT       GetCameraPoseResponse(Pose)
16    HT -> ARE      TrackObjectResponse(1,Hand, PoseHyp1, PoseHyp2)
17    ARE -> HT      StopReportingOnHypothesis (1, PoseHyp1)
18    ARE(GR)        RecogniseGesture
19    ARE -> GR      AutoUpdateGesturesForHandObject(*HT ,1, Hand)
20    GR(HT)         TrackObject
21    GR -> HT       TrackObject(1, Hand, PoseHyp1)
22    HT -> GR       TrackObjectResponse (1,Hand, PoseHyp1)
23    HT -> ARE      TrackObjectResponse (1,Hand, PoseHyp1)
24    GR -> ARE      UpdateGesturesForHandResponse(Gesture1)
25    ARE -> HT      CancelTrackObject(1)
      // object 1 is further tracked because of service request of the GR!
26    ARE(HT)        Cancel TrackObject
27    ARE(ORG)       DetectProximity
28    ARE -> ORG     AutoUpdateObjectsNearTrajectory(*HT ,1, Hand,
                                                     PoseHyp1, 1m)
29    ORG(HT)        TrackObject
30    ORG -> HT      TrackObject(1, Hand, PoseHyp1)
31    HT -> ORG      TrackObjectResponse (1, Hand, PoseHyp1)
32    HT -> GR       TrackObjectResponse (1, Hand, PoseHyp1)
33    ORG(REC)       Recognition3D
34    ORG -> REC     RecogniseFromCamera(PoseHyp1)
35    REC(IS2)       ImageServer
36    REC -> IS2     GetShmImage()
37    IS2 -> REC     GetShmImageResponse(ShmKey)
38    REC(PS)        CameraPoseServer
39    REC -> PS      GetCameraPose(2)
40    PS -> REC      GetCameraPoseResponse(Pose)
41    REC -> ORG     RecogniseFromCameraResponse(2, Pose1, Hyp-CD,
                                                     Hyp-Player,
                                                3, Pose2, Hyp-CD)
42    ORG -> ARE     AutoUpdateObjectsNearTrajectoryResponse(
                                  *HT ,1, Hand, PoseHyp1, 2, Pose1,
                                  Hyp-CD, Hyp-Player, 0.4m)
43    ORG(MS)        ModelServer
44    ORG -> MS      GetModelTypes(CD)
45    MS -> ORG      GetModelTypesResponse(Wireframe)
46    ORG(OT)        TrackObject
47    ORG -> OT      TrackObject(2, CD, Pose1)
```

```
48    OT(MS)          ModelServer
49    OT -> MS        GetModel(CD, Wireframe)
50    MS -> OT        GetModelResponse(CD, Wireframe, Model)
51    OT(IS3)         ImageServer
52    OT -> IS3       GetShmImage()
53    IS3 -> OT       GetShmImageResponse(ShmKey)
54    OT(PS)          CameraPoseServer
55    OT -> PS        GetCameraPose(3)
56    PS -> OT        GetCameraPoseResponse(Pose)
57    OT -> ORG       TrackObjectResponse (2, CD, Pose1)
58    HT -> ORG       TrackObjectResponse (1, Hand, PoseHyp1)
59    HT -> GR        TrackObjectResponse (1, Hand, PoseHyp1)
60    ORG -> ARE      AutoUpdateObjectsNearTrajectoryResponse(
                                        *HT ,1, Hand, PoseHyp1, 2, Pose1,
                                         Hyp-CD, 0.4m)
61    ARE -> AP       ProduceConceptFunctionResponse(Activ1)
62    …
      .
100   AP -> HMI       CaptureExemplarResponse()
```

B.6 References

[1] Schlegel C., Wörz R. „The Software Framework SmartSoft for Implementing Sensorimotor Systems”, IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS '99, 1610-1616, Kyongju, Korea, October '99

[2] www.orocos.org

[3] Henning M., Vinoski S. (1999). “Advanced CORBA Programming with C++”, Addison-Wesley, Massachusetts