IB - An Information Bus: A Multilayered Information Base Interface for Remote Applications

Bjørn Skjellaug

Dept. of Information Technology,

Centre for Industrial Research (SI),

P.O.Box. 124 Blindern, 0314 Oslo 3, Norway.

email. addr: skjellaug@si.uninett

Abstract

A framework for an information bus (IB) is presented. The IB integrates information bases and remote application systems. Within the framework we present an outline of an integration methodology, which includes integration analysis and conceptual specification of an information base interface. The analysis highlights and structures the strategic and technical aspects of an integration. The aim of the conceptual step is to describe the information in a formal and abstract way. This specification formalism is based on object-oriented methodology (as objects and operations). An example of application of the IB is given.

Keywords and phrases: Framework for an information bus, integration methodology, integration analysis, object-oriented conceptual specification, interfaces, information bases.

1. Introduction

The nineties will be a challenge to the hardware and software vendors. In the area of system integration, the market will demand standardized products and solutions. Still, products are developed with "ad hoc" or so called "de facto standard" solutions. These products are not always directly integratable with other (hardware/software) systems (based on standards), an will increase the integration costs significantly.

In the present decade there has been a strong coordination between ISO and CCITT[1] (see endnotes) in standardization work on software and communication. This has in turn resulted in national and international activities (ref. [1],[2],[3],[4],[5] and others), which promote profiles based on ISO and CCITT standards and recommendations respectively.

Still, we have the feeling that, with respect to integration, there is a gap between the application development and communication development communities. Techniques and methods that could help to specify the integration requirements are missing.

This paper presents a framework as a study of the complex and heterogeneous environment that developers and programmers face when extending an existing environment to include **both** automatic exchange and automatic transfer of information between systems. The application area addressed in this paper is remote use of information bases, which is relevant within office automation and CIM[2]. Different bottle-necks exist, and must be considered and analysed before the realization of the integration. This includes aspects as "openness" of the application systems as well as types of services provided by the communication systems, denoted in this paper as horizontal and vertical integration aspects respectively.

The objective is to present an integration methodology within the framework. This methodology consists of two parts. First an integration analysis that highlights and structures the differents levels of the IB which affect the integration. The analysis is given a general and informal (natural language) description and is the basis for an integration process. Secondly, an outline is given for a more technical description of the information base interface. This part of the methodology is based on an object-oriented approach (but not limited by it) and serves as a common platform for the conceptual level of the integration process, i.e. the conceptual specification.

This paper is structured as follows; Chapter 2 introduces an informal description of the strategic and the technical aspects of the framework. Chapter 3 gives the formalized methodology for how to form a conceptual specification of an information base interface. Chapter 4 gives an example of an application (the example is related to the DIMUN[3] project [6], project no. 1039 within the EEC RACE program), and finally chapter 5 contains a discussion of the framework .

2. A Framework of An Information Bus

The framework describes all parts of the integration process, some more detailed than others. This chapter focus on the overall problem of an integration process as an introduction to the domain it self. We have already mentioned that we face a horizontal and a vertical process when we want to integrate different sites, connected by networks and network services.

This chapter focuses on these two parts. As a starting point we say that systems before the integration are autonomous, and after they are integrated as a whole by the IB. In this context the horizontal integration (exchange) is concerned with the information representation and presentation. On the other hand, the vertical integration (transmission) is concerned with how the information is transferred.

The information bus (IB) is a multilayered information base interface. Therefore, the IB will be described by the different layers that are involved in such an architecture. The layers and their interconnection is the basis of the framework. The framework is intended to be a supplement within the strategic and technical decision making.

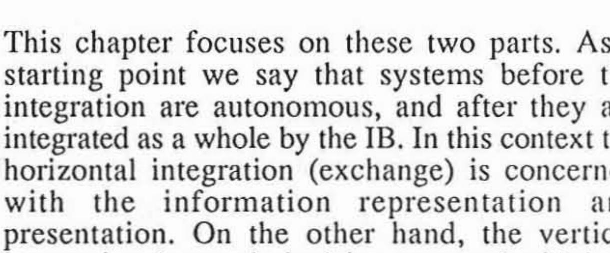


Figure 1 - The Integration Process: before the systems are autonomous, the integration has opened the systems. Indicated by the plus sign of A^+ and B^+ after the integration.

The role of an IB is to integrate different autonomous systems. Figure 1 illustrates the result of an integration process. The autonomous systems A and B are marked A+ and B+ after the integration, this illustrates that the systems have been "opened" in some way or an other as a result of the integration. In figure 1, the integration solution has solved both the exchange and transfer problems, i.e. horizontal and vertical problems respectively.

2.1 Two main integration aspects

We have mentioned two main aspects which both are essential for the possibility of integration. We will describe both of these aspects in this section. First we give a little introduction to the concepts. They differ mainly in their nature of information handling. The first is capable of a semantic manipulation of the information, the second is only capable of a syntactic manipulation of the information.

The basic difference can be summarized as followed: The information can be viewed as an object, where the semantic manipulation knows both the interior and the exterior of that object. The syntactic manipulation knows only the exterior of the object. In other words, the information is encapsulated in that object.

Meaningful manipulation is not possible if an operation does not know the interior definition(s). This leads us to choose an object-oriented approach for the methodology presented later in this paper.

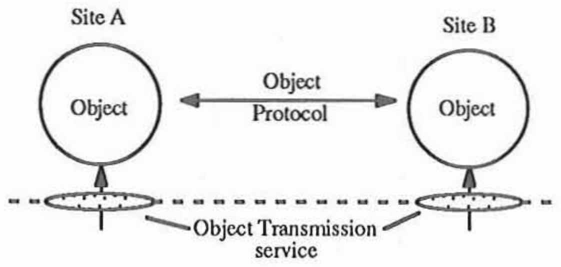


Figure 2 - Two integration aspects

In figure 2, the two integration aspects are illustrated by means of the horizontal and vertical concepts - object protocol and object transmission service - which we define below.

2.1.1 Horizontal Integration

Horizontal aspects in this paper means the semantics of the information and the logical solutions which are closest to the application level. Beware that with an application we mean both the information bases and the connected end systems.

The term horizontal is chosen because the systems that are able of interpreting the information are on the same semantic level.

We may chose among four types of logical solutions for an integration (or an combination of these.) They are:

- common information base
- common user interface
- standard information formats
- common information directory

Common Information Base: All involved systems use the same pool of information that is relevant for use by the systems.

Common User Interface: All systems are available through one uniform user interface. A user does not need to download and upload the systems each time he or she is going from one system to an other.

Standard Information Formats: All systems have pre- and post-processors to translate between native and neutral formats.

Common Information Directory: All systems may use this directory to get information about information, i.e. meta information.

For all four types of integration strategies there are objects which are exchanged. These objects must have a meaning for the applications or users.

Note that in three of the strategies, the transmission (vertical) aspect is already implicitly included. The one that is missing this aspect is the standard format. This gives only a format or a language for which native formats can be translated to or from a neutral one.

As an example we take an ODA[4] or an EDIFACT[5] document, here called an ODA or EDIFACT object. The interior of the object is the specific structure and contents of the document. If the object is exchanged from one system to an other, both systems can, if desired, interpret the information according to the agreed format and language.

We say that both systems know the semantics. The description of the semantics is included in an object protocol definition. This object protocol is then the horizontal integration solution.

2.1.2 Vertical Integration

The vertical integration is the transmission of the information without any concern on what is transmitted. We may say that the transmission is some kind of an Object carrier, which carries the information object from one site to an other.

As in the example mentioned above, where we have an ODA or an EDIFACT object, we need a transmission of this object, i.e. an object transmission service. For both ODA and EDIFACT objects there are, to some extent, specified services for transmission of such objects. These services know the basic syntax structure, and are able to path the objects as messages from one site to an other. The service definitions (as well as the object protocol standardization) is done within ISO and CCITT.

The object transmission service is the vertical integration solution.

2.2 The IB Architecture

In the previous section we have given a general description of the two integration aspects of the Information Bus. In this section we will describe the architecture of the IB. We focus on each of the layers, and show the sequence of steps that must be undertaken as parts of the integration analysis.

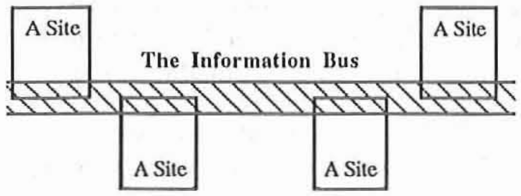


Figure 3 - The Information Bus with different sites connected.

In figure 3 the IB (the hatched area) is illustrated as the communication bus which provide facilities for exchange and transfer of information. Each site represents one or more activities that are integrated with other activities (locally or not) through the IB. Thus, the IB can also be a concept for both the internal integration of activities, as well as integration of widely distributed activities, see figure 4.

The IB does not intend to support distributed mechanisms such as those provided by distributed DBMS. It rather gives a strategic and technical basis for a requirement specification of the system integration analysis. This involves requirements on how the object protocol represents and structures information as well as the required object services for transfer of the desired information.

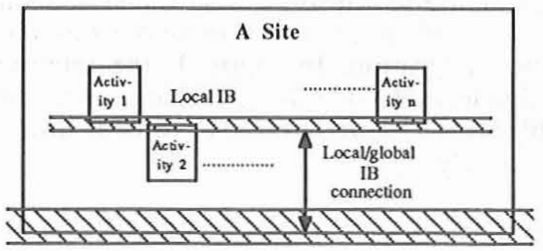


Figure 4 - "Local" and "global" IB, and the interconnection of these two.

2.2.2. The IB layers

We shall describe the strategic parts of the integration analysis, divided according to the layers of the IB. All layers of the IB are strategically important within the decision making.They all influence the functionality of the IB. Still, the underlaying layers of an IB are transparent to the applications or the users. But they are not transparent for those who perform the integration analysis. Each layer represent a step of the integration analysis.

In figure 5 the IB is presented with three main layers, each one is described below. Note that "an application" will here mean either the information base(s) or the remote application(s) connected with the information base(s) through the IB.

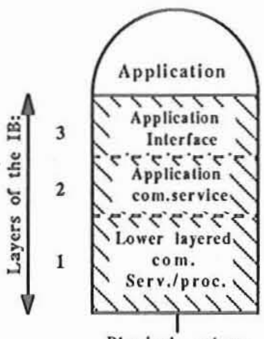


Figure 5 - IB´s three layers (hatched area).

The lower layer of the IB :Contains at most six layers corresponding to the 7 layered ISO´s Open Systems Interconnection (OSI) reference model [7]. In the IB they are: the presentation, session, transport, network, link and physical layers from top to bottom. The problem with this layer of the IB is that network (and transport) services may support different solutions from one installation to an other. They are named Connection-mode Network Service, CONS, and the Connection-less-mode Network Service, CLNS. The CLNS is described in an addendum [8] to the OSI reference model. There are gateways which can interconnect networks of different types, but these are not at all standards. The CLNS was first defined by DARPA[6], and are supported by most of the computer vendors, known as the TCP/IP network.

The application communication layer: Corresponds to what the OSI names as the application layer, layer 7 of the reference model. This layer provide communication services to the application systems. A DARPA protocol also contains this layer, but does not include the layers 5 and 6. These services are also named value-added network services (VANS), e.g. filetransfer (FTP, FTAM), mail systems (MHS, X.400, EAN), MAP, TOP, etc..

The crucial point at this level of an IB rises two possible decisions concerning the VANS:

• The first concerns whether the organization is connecting its own IB to a global IB. Then the local IB must at least support (some of) the services which is supported by the global IB.

• The second reflects only an internal installation, from scratch or as extension, of a local IB. Here the question is: how can one obtain more and better functionality between applications? Some services may in some cases give more overhead because of a lot of human interaction in the point to point transmission, e.g. performing file transfer by a mail system. Therefore, the organization will be better off with a critical analysis on the possible services to choose.

The application interface layer: Is illustrated in figure 1 as the plus signs of system A^+ and system B^+ after the integration. This interface specifies the special purpose or the extended functionality of the application. The openness of a system decides if a system can be integrated with some other system(s) or not [9]. The decision whether the system is "open" or not can be investigated according to the following criteria:

• Does the system have functions/commands to external system(s)?
• Does the system have operating system commands?
• Does the system support standard exchange formats?
• Does the system support some other known format(s)?
• Is the internal information format described in the manual, and may the
information be read in some way or an other?
• Is the system able to read and to manipulate information from
an external system/device?
• Does the system support some kind of extension for user specific commands?
• Does the system support any communication services?

At this stage the analysis only concerns the application requirement input. And these requirements will be described in terms of services and information. When an information base is considered, the services are on the form **store, retrieve** and **update.**

The framework support these services as the basic services to the applications. Be aware that these services have an additional description, and should not be mixed with typical DBMS operators, [10], with the same names.

• **STORE (INSERT):** This service acts as follow: The user has sent a write request for some information object to be stored into the information base, i.e. a new entry. The service will then call upon the information base store mechanism, which store the information object if OK.

• **RETRIEVE:** This service acts as follow: The user has sent a read request for some information object. The service will then call upon the information base retrieve mechanism, which returns the desired information object if OK.

• **UPDATE (MODIFY):** This service acts as follow: The user has sent a request for an update of some entry in the information base. The service will then call upon the information base update mechanism, which update the entry if OK.

The framework does not consider how the user, here the remote application, actually performs a request. But the framework intend to address the connection, which the IB serve, between an information base and a remote application. The application interface virtually connects the information base(s) and the remote application(s). That is the horizontal connection named as the object protocol, e.g. ODA and EDIFACT object protocols.

The application interface layer of the IB is now described by three possible services. At this stage we may for simplicity say that the above services include the object protocol and the object transmission services, i.e. both the horizontal and vertical integration aspects. We will later in this paper show that the services on this layer in fact are defined with these aspects and give a more clear distinction between different objects handled by the specification methodology. (So fare we have distinguished between two types of objects.) The conceptual specification, based on the analysis, must include additional properties of each service. These properties are:

• Functionality, described above.
• Primitives, what kind of system specific mechanisms are used, both input to and output from application and communication systems.
• Parameters, what type of information is handled, both input to and output from application and communication systems. The returned result is positive or negative, i.e. success upon request or not.

A more technical and detailed description of the application interface layer is given in chapter 3.

2.3 The Client - Server concept

In the previous section the application interface layer of the IB was described. This interface gives the description of the conceptual specification (based on the object protocol definitions) of the information the remote application and the information base may exchange. This leads to view the remote application and the information base as the client and the server respectively, exchanging information objects.

There may be different ways to implement the interaction between a client and a server. The best known mechanism of such an interaction is the "Remote Procedure Call" (RPC), first described in [11], based on ordinary program-language procedure calls. Here, the main idea is that it is transparent to the user or the calling program/process whether the execution is performed locally or transferred to a remote process.

All client-server interactions (of the specific application area in this paper) can be viewed as illustrated in figure 6.

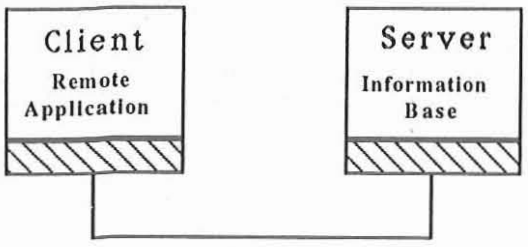


Figure 6 - Client and Server interaction

The server-client concept within the framework is chosen because it provides the transparency for client(s) when interacting with the server. The object-orientation may well be applied as a conceptual extension to the server-client concept. A wider description of the client and server communication and interaction is found in [12].

3. The Methodology

The previous chapter presented an informal description of the IB. In this chapter we first present the integration analysis and then an object-oriented methodology for the conceptual specification of the application interface.

3.1 The Integration Analysis

One of the objectives of the framework was to get a better understanding of the complex environment faced by system integrators and that the integration analysis should support the decision making. The previous chapter has described the layered IB and the crucial points of the integration process.

The output of the analysis is the strategic and technical decisions (including the network interconnection, VANS and application interface.) The analysis takes into account each layer of the IB, but the most critical bottle-neck is the "openness" of the end-systems, shown as the application in figure 5.

The three different layers of the IB require four major analysis steps of the integration process. A natural language description of the analysis steps of the methodology is given below:

• Are there different networks in the environment and is it possible to connect them?
Reflects the lowest layer of the IB.
• What types of services (and which VANS) do the networks support ?
Reflects the second layer of the IB.
• What kind of application requirements are there?
Reflects the upper layer of the IB.
• Is the conceptual specification of the IB's application interface possible?
Reflects the coordination of the previous steps.

A breakdown of the analysis into tasks can be done with help of introducing a flowchart like symbolism as shown in figure 7.

□	Identification of Task
◇	Task Decision
▢	Input to/Output from Task
→	Flow
- - -	Coordination between tasks
Iteration?	Backtracking/Termination

Figure 7 - Symbols of an analysis flowchart

The integration analysis steps can then be described with these symbols as shown in the diagram of figure 8. In figure 8, there are four steps indicated with the task symbol. Each of these tasks will in turn be described by other diagrams. That is, a breakdown into input/output, task and flow symbols etc.

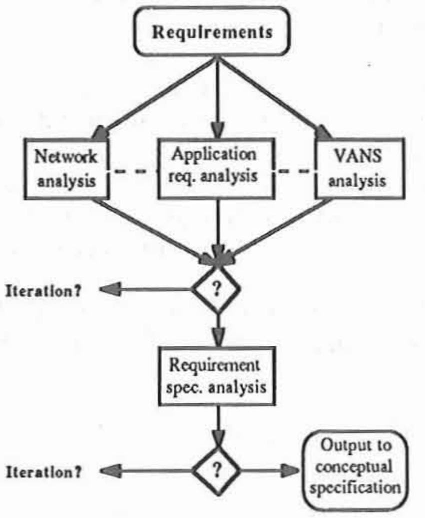

Requirements
Network analysis
Application req. analysis
VANS analysis
Iteration?
?
Requirement spec. analysis
Iteration?
?
Output to conceptual specification

Figure 8 - Steps of the integration analysis

In figure 8, the question marks indicate each decision to be made. That is, should the next task(s) be performed. If not, one must decide whether the analysis should be continued by some backtracking to the previous task(s) or the termination of the analysis. This is marked **Iteration?** in the diagram of figure 8. An iteration may as well include an evaluation of the requirements. The output of this analysis may be used for the conceptual specification. (The output from each step is implicit in figure 8.)

The integration analysis is a bottom-up analysis. (On the other hand, the conceptual specification is top-down.) That is, all the decisions are based on separate parts of the IB, and the analysis ends up with a common conclusion based on the part analysis decisions. Below each part of the analysis are described with input/output, and task decisions. A breakdown of tasks to any level of detail within the analysis is possible. A task then gives rise to a more detailed analysis part. We only show the main task(s) of each part analysis in this paper.

3.1.1 The Network Analysis

This step has input requirements like: which sites should be interconnected, which networks are involved, available computers, network and transport service solutions, missing or available gateways between networks, etc.

The diagram of the network analysis are shown in figure 9.

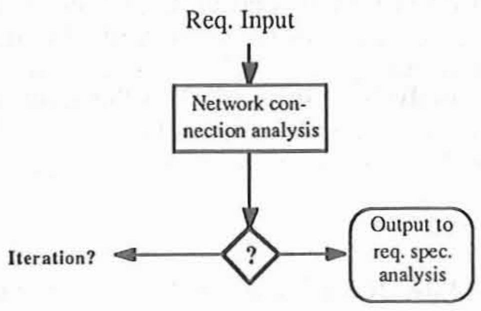


Figure 9 - Network analysis

The task of the network analysis is to decide if the involved networks and computers can cooperate in an environment. The output from the network analysis is used in the requirement specification analysis which ends up with a decision if it is possible to integrate the desired systems.

3.1.2 The VANS analysis

The input requirements to this part analysis are of the type; types of VANS supported by the different networks and computers. The effort here is to find out if the different VANS provide the same functionality and may they be used on the different computers.

A diagram for this analysis step will be the same as that shown in the figure 9. The output is input to the requirement specification analysis.

3.1.3. The Application requirement analysis

This analysis has input as follows: description of the application(s) requirements, information and system (operation) requirements.

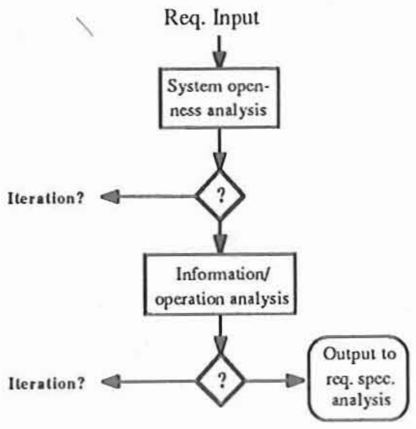
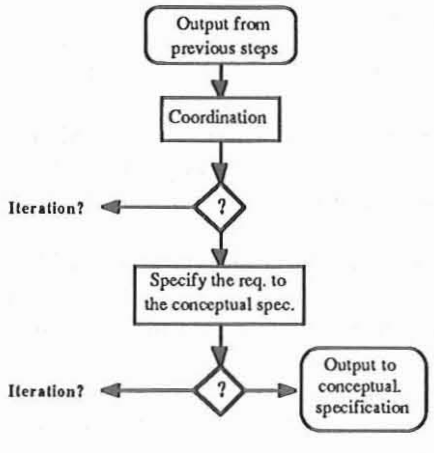

Req. Input
System openness analysis
Iteration?
?
Information/ operation analysis
Iteration?
?
Output to req. spec. analysis

Figure 10 - The application interface analysis

The first task of this analysis part is to find out if the end-systems are open, in the terms of the criteria described in section 2.2.2. The second is to analyse the information and the desired operations and give an output on this. The output of this analysis will be used in the requirement specification analysis. We will later in this chapter give a detailed description of the application interface.

3.1.3. The Requirement Specification Analysis

The requirement specification analysis is based on the previous part analysis outputs. The main effort here is to decide whether a coordination of the analysis outputs give the basis for an integration of the systems. The output of this part analysis will be used in the conceptual specification of the application interface which we present a methodology for in the next sections. This part analysis is shown in figure 11.



Output from previous steps
Coordination
Iteration?
?
Specify the req. to the conceptual spec.
Iteration?
?
Output to conceptual specification

Figure 11 - Requirement specification analysis

The output of this analysis gives an informal input to the conceptual specification, which is the next part of the methodology. The input must be on the form: types of information (see section 3.2.2), operations on the information and transmission services.

3.2 The Application Interface

In this section a more detailed description of the IB´s application interface is given. All details will as far as possible be described in terms of object-oriented concepts.

3.2.1 The Functional levels

The functional levels of the IB´s application interface is the mapping between the application services and the communication services.

There are three levels of the application interface, see figure 12.

- The application service level. Supports mapping to and from the information base mechanism.
- The application /communication service level. Supports mapping between the application and communication service levels.
- The communication service level. Supports mapping to and from the VANS.

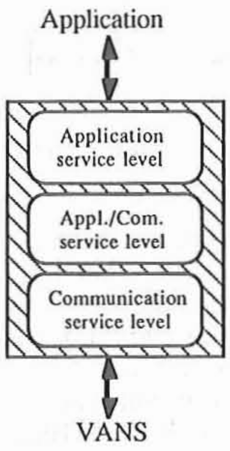


Figure 12 - Levels of the application interface

In the application interface the services for the three levels are:

- Application service level:
  * *Store*
  * *Retrieve*
  * *Update*

- Application/Communication service level
  * *MapUpwards*
  * *MapDownwards*

- Communication service level
  * *RequestSend*
  * *RequestReceive*
  * *RequestReturn*

The list of services above is not complete, it indicates the main services of the application interface.

3.2.2 Information objects

Information objects are what the systems exchange. The information object definitions contain the user or application specific conceptual scheme of the Universe of Discourse (UoD). The UoD is the part of the real world we are modelling. The methodology does not give a conceptual schema description in the sense of a traditional database schema found in [10].

An information object is containing the information and the representation definition, explicit or implicit, of that object. The representation can according to a predefined model be included in the class definition of this type. Each information object has a class definition.

The information representation could use many types of formats or models, we have already mentioned two: ODA and EDIFACT. Other formats are also possible. Information objects may also include operations such as conversion between an application specific representation and the communication specific formats.

Information objects could be composite or complex objects, consisting of any level of object abstractions (ref. [13], [14]). The lowest level of information abstraction is elementary data (or elementary information objects), objects which are not described by means of other information objects. The elementary information objects could be modeled by use of traditional data models such as the relational or network model ([10]), or the entity-relationship model [15]. The complex or composite information objects are in this context aggregates of other objects. For the methodology we have two levels of abstraction for the information objects. The first level is the identification of the objects. The second is the modelling of the objects by means of existing data modelling techniques.

Because we let the information objects be described by means of different data models, there must at least be two constraints which are satisfied. The information objects which are composite or complex objects must fulfil the following two constraints [16];

- **Completeness** - An information object is complete when it represents aspects expressed in the requirement specification analysis output. Thus, every composite or complex object refers to concepts that appear in the elementary definition(s).
- **Correctness** - An information object is correct when it uses the concepts of the model(s) to represent the output of the requirement specification analysis.

3.2.3 Object description

In section 2.2.2 three main properties for a service were described, the functionality, the primitives and the parameters (attributes).

The Service type is the generic type of all objects described in the application interface. An object is an abstract object, and the technique permits describing:

1) The characteristics of objects which correspond to static aspects of the application interface.
2) The handling of these objects which correspond to the actions of the various services of the application interface.

The abstract objects described in a specification of the application interface will improve the understanding of the role and the actions of the IB´s application interface. This leads to two partial descriptions for each object, or abstract entity:

1) the characteristics of this entity, and
2) the operations of this entity.

From the above we have divided into to three types of distinctions. The first is the functional description, called the services. The second describes the characteristics (semantics) of an object, also named as the object protocol. The third is the set of operations which perform the transmission, also named as the object transmission services. Note that we distinguish between operation on the information and operation for transmission of information.

The first of these distinctions is a generic type which we shall see includes both the object protocol and the object transmission.

3.3 The Object-Oriented Implication

The idea of an object-oriented approach is the mapping of a human´s understanding of a real-world phenomena into a computer-based description [17]. That is, the mapping of the UoD into some sort of a conceptual scheme. The use of an object-oriented approach makes the mapping more in terms with real-world phenomena and concepts. A real (or imaginary) part of the world (phenomena and concepts) behaviour may then be simulated as a physical model [18].

The object-oriented approach presented herein is not based on any specific object-oriented language or method. The way of introducing it here is the top-down mechanism and its "close to real world" static and dynamic structuring. The static structuring will be focused in this paper. For dynamic structuring the use of evaluation net description can be applied. An evaluation net is an oriented graph using three types of nodes (states, requests and transitions), and can be a supplement to the object-oriented approach.

3.3.1 An Object-Oriented Approach

The methodology applied uses the following basic object-oriented properties [18], related to specification (and design) rather than programming techniques:

- **Phenomena aspects**: The physical matter that we identify in the UoD. These phenomena can be given properties. And transformation of these properties are possible.

- **Concepts aspects**: For the modelling aspects of phenomena it is necessary to use abstractions or concepts. A concept notation has the following elements:
  - * *Name*: Denoting the concept.
  - * ***Intention***: The properties characterizing the phenomena covered by the concept.
  - * ***Extension***: The phenomena covered by the concept.

- **Abstraction**: Is the process of creating the concepts. Three main abstraction mechanisms are shown, they all focus on similar properties of phenomena:
  - * ***Classification***: Phenomena covered by the same concept.
  - * ***Aggregation***: Concepts defined by means of other concepts.
  - * ***Generalization***: Concepts may be organized in a classification hierarchy.

In classical object-oriented notation a phenomena is denoted as A Class or A Generic Class, the latter as the generic classification abstraction of the former. A Class is described by attribute types, i.e. properties of the phenomena, and (sequence of) actions, i.e. the transformation of phenomena properties.

An Object is an instance of A Class. The characteristics of the instance is that a class is instantiated by means of Class and Instance Attributes issued from the generic class. However, this is outside the scope of this paper.

A class can further be a subclass of an other class (denoted superclass), where the subclass inherits the properties and transformations from the superclass. This is called generalization.

Where a class is described by means of other classes, the class is an aggregate of the other classes. The aggregates only concern the information objects of this methodology.

3.3.2 Object - Syntax Description

A Generic Object is defined as followed:

Generic Object: (name of the generic class of the object)

<u>Example:</u>
Generic Object: SERVICE
(generic object associated to a service)

A Class is defined as followed:

Object: (name of the class of the object)

<u>Example:</u>
Object: SERVICE (class associated to a named service)

A Class reference is defined as followed:

```
Object: (name of the class of the object)
        {
          Attribute: ^(name of referenced class)
          Key-Attribute: (key-attribute name)
        }
```

A key attribute identifies, uniquely, an object, a class or a generic class

A SubClass is defined as followed (single inheritance):

```
Object: (name of the class of the object)
   {
   Attribute: (attribute name)
   Attribute: Inherit from (name of superclass)
   }
```

A Class method is defined as followed:

```
Object: (name of the class of the object)
   {
   Functionality:   (* a description of the object's
                         functionality  *)
   Attribute: ^(class name) | (attribute name)
                    ( default values optional )
```

```
    ..........
    Primitive: (* types of primitives to be used*)
        [
        Condition:
            (attribute name) = ^(value object class name)
        (primitive name) (Input, Output)
        Input:
            ^(class name) | (attribute name)
            .........
        Output:
            ^(class name) | (attribute name)
            .........
        Do
            #
            (* actions described in terms
                of imperatives *)
            IF (Condition)
                (*imperatives*)
            ELSE
                (*imperatives*)
            #
        ]
    }
```

The different parts of the object are enclosed as followed: object body by {} brackets, the object primitives by [] and primitive imperatives by ##.

The characteristics of the object is described by a number of **Attributes** and the operations on objects are described by a natural language (**Functionality:**) and service primitives (**Primitive:**).

The primitive will only be executed if the **Condition:** holds, i.e. true. However, the condition is optional. The condition refers to a single attribute of this class. The value object is a specific object containing the possible values which can match the attribute value.

Note that the output and input parameters indicates which attributes or other objects are transferred or received by this object definition. This is done by pathing a message (contents of an attribute or reference to an object) or receive a message.

The methodology uses the common terminology of objects, attributes and methods (primitives), and type notation of such.

3.3.3 Examples

To exemplify the use of the object-orientation part of the methodology for the conceptual specification we will use the services for an information base application listed in section 3.2.1, and relate it to the example of section 2.1.

In section 3.2.3, we said that all three levels of the application interface are defined as objects instantiated of an generic type (see *GenericService* below). Each of these objects are in turn superclasses for the service listed in section 3.2.1, e.g. *Store* is a subclass object of the superclass object *ApplicationService*, and therefore inherits all its properties and primitives.

A few examples on the definitions of some application interface objects are given. These examples show how the methodology incorporates the analysis output directly into the definitions of objects. Where the different parts of the analysis output such as functionality, primitive and information object descriptions are formalized in the object definitions.

Example 1:

```
Generic Object: GenericService
    {
    Functionality: (* Serve the needed support of
                        mappings in the application interface *)
    Class Attributes: .......... (* class specific *)
    Instance Attributes: ...... (* instance specific *)
    }
```

Example 2:

```
Object: ApplicationService
    {
    Functionality: (* provide an action upon request
                        to the information base and returns
                        the result of the request*)
    Attribute: Status (default error value)
    Attribute: ..........
    Primitives:     (* Store, Retrieve or Update
                        at this level*)
        [
        Info (Output)
        Output:
            Status  (* returns a status value *)
        Do
            #
            Status :- ^InfobaseStatus
            MapDownwards (Status)
            #
        ]
    }
```

Example 3:

```
Object: Store
    {
    Functionality: (* call upon the information base
                        store mechanism if ok *)
    Attribute: Inherit from ApplicationService
    Attribute: Req.Name
    Attribute: .........
    Primitive:      (* information base store *)
        [
        Condition:
```

```
        Req.Name = ^InfobaseUserlist
      InfoStore (Input, Output)
        Input:
          ^Information (* object containing
                              info to store *)
        Output:
          Status (* inherited from
                        ApplicationService *)
      Do
        #
        (* calls upon the information base routine and
        unpacking of the Information object *)
        IF (Condition)
          InfobaseStore (^Information)
          Info (Status) (* return status of call *)
        ELSE
          MapDownwards (Status)
                (* return default status *)
        #
      ]
    }
```

From the examples above the generic object *GenericService* definition is straight forward. The *ApplicationService* has a more detailed definition. This service is defined by means of a Class Instance of the *GenericService*. The *ApplicationService* has a *Status* to return after termination. All services which are subclasses of the *ApplicationService* therefore inherit this *Status* assigned by the primitive *Info*.

The *Store* service is a subclass of the *ApplicationService*, and has inherit its description. Attribute *Req.Name* of *Store* must be assigned a user name before this object will execute its primitive *InfoStore*. In this case the user name is for the authorization mechanism of the information base interaction, i.e. the user name of the remote application. The input to *InfoStore* is an information object called *Information*. This object contains the specific information to be stored. The information is defined by the identification shown in the example above and with some basic data modelling technique referred to in section 3.2.2.

As an illustration the information object *Information* could be an ODA or an EDIFACT object, which include the structure and contents of a specific ODA or EDIFACT document. On the other hand the information object could be simply a SQL like INSERT statement ([10]), which include both information and operation.

The examples above are describing objects at the server site. The same set of objects must also be defined for the client site. But, some differences exist, such as a request upon the store service. The client site will *MapDownwards* the *Information* the server site will *MapUpwards* the *Information*. The same is true for the *RequestReceive* and *RequestSend* services. However, the information object definitions must respect the constraints described in section 3.2.2.

From the above definitions we may show how the sequence of actions are taken place for a client store request.

Actions at client site:

```
  Store (^Information)
      (the information object store request)
  MapDownwards (^Information)
      (mapping between levels)
  RequestSend (^Client-ident)
      (transmit the identification object first)
  RequestSend (^Information)
      (transmit the information object)
```

Corresponding actions at the server site:

```
  RequestReceive (^Client-ident)
      (receives first the identification object)
  RequestReceive (^Information)
      (receives the information object)
  MapUpwards (Req.name:- ^Client-ident)
      (mapping between levels)
  MapUpwards (^Information)
      (mapping between levels)
  Store (^Information)
      (store if condition holds)
```

4. The DIMUN project

The DIMUN project is a project in the Usage group within the RACE program. The project is focusing on applications of Integrated Broadband Communication (IBC) and utilization of these in distributed manufacturing. The project has broader objectives than those addressed by the framework and the methodology described in this paper [6]. DIMUN is divided into several workpackages, and one of these concerns integration of information base(s) and remote applications of the manufacturing process. The methodology presented in chapter 3 is based on the need of a tool to specify such an information base interface.

In this workpackage the analysis and requirement specification of the information bases and interfaces were executed in ´88. The outcome was an identification of information and interfaces of such an environment. The analysis did not concentrate on the lower layers of an IB, only on the application layer. A brief description of the results and future work are presented below.

An information base is grouped into three main logical units [19]:

• **Customer DB** - contains information from/to/for customers, e.g. customer name, address, request, offers, orders, public relation documents, etc..

- **Product DB** - contains information of products, e.g. product structure, technical drawings, design data, calculated data, etc..

- **Manufacturing DB** - contains information about the manufacturing process, e.g. structure of the distributed manufacturing process, planes/schedules, monitoring information, reports, etc..

The connection between a remote application and a DIMUN information base is shown in figure 13.

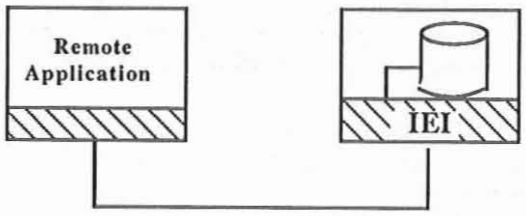


Figure 13 - A remote application - DIMUN information base interaction

For each site in the distributed manufacturing process there may be one logical information base. Each one is interfaced with the communication systems with a so called Intelligent Enterprise (Server) Interface (IEI). The levels of the application interface in chapter 3, are also present in the IEI. The information base interface is one service of the IEI and is described by the methodology presented in chapter 3. Thus, the IEI is more than an information base interface, it is a concept which interfaces all the services of the enterprise [19].

The first prototype of the IEI will be implemented summer/autumn '89. The IEI prototypes will be installed by the pilot partners in other workpackages of this project.

The IEI prototype is planned to exchange and transmit technical information (CAD drawings and images) as well as business like information (orders and offerings) between different partners of the project.

For each type of information there is three objects to be defined and specified. The first is the application service, which gives the basis of handling of the information. The second is the information objects (i.e. object protocols) listed above for the three logical units. The third is the transmission (i.e. object transmission service) which in DIMUN are ordinary VANSs.

5. Discussions

The framework presented in chapter 2 gives an informal description of the complex environment faced by integrators, when it includes information exchange and transmission of information. We have distinguished between these two aspects and showed that the aspects reflects two areas within integration. That is, the horizontal and vertical aspects. Therefore, we also include both aspects within the framework. They are both part of the methodology, first in the analysis and later in the conceptual specification. To our knowledge there are no other framework or methodology that addresses both the aspects of integration of heterogeneous systems.

These two topics are not necessary a twofold of the same integration process. In the former case we may only need some kind of standard exchange formats such as IGES[7], ODA or others, and corresponding pre/post processors. In the latter case it is a matter of an automatic and physical transmission of the information. This is done without concern to the information semantics, as in most of OSI's application services.

The methodology presented in chapter 3, includes different data models for the definition of the information object contents. This part of the methodology makes it possible to describe the integration of existing heterogeneous systems.

The methodology is used to specify the static parts of the application interface. It has also improved the understanding and structuring of the Intelligent Enterprise Interface, and will be used as the technical documentation of this interface.

Acknowledgements: The framework presented here is mainly a result of the author's participation in the DIMUN project. A thank to all partners for interesting discussions, as were comments by Lise Arneberg and Heikki Hämmäinen on an earlier draft, and by Per Anton Fevang on the last version of this paper.

References:

[1] Towards a Dynamic European Economy, Green Paper on the Development of the Common Market for Telecommunications Services and Equipment; Commission of the European Communities, DG XII, May 1987.

[2] "UK- GOSIP, The Government Open Systems Interconnection Profile", HMSO Publications Centre, London, ISBN 011 330 5176 and ISBN 011 330 5184.

[3] K.Å. Bringsrud, E. Mjøvik and T. Grimstad "National Plan for OSI-net in Norway", Norwegian text, NCC-note DTEK/08/88, Norwegian Computing Centre, Oslo, Norway.

[4] "ISONET-S", a IT4-project in Sweden Wolf Arfvidson, Stadskontoret, Stockholm, Sweden.

[5] "COSINE, Cooperation for Open Systems Interconnection Networking in Europe", Cosine Policy Group, Commission of the European Communities, DG XIII-A2.

[6] "Analysis of Status quo and Communication Requirements", DIMUN, project no. 1039, Deliverable D1-2 to RACE Office, Commission of the European Communities, DGXIII-F, Telecommunication, Information Industries and Innovation.

[7] Open Systems Interconnection - Basic Reference Model, ISO 7498

[8] Addendum to Open Systems Interconnection - Basic Reference Model, ISO 7498/DAD 1

[9] Skjellaug Bjørn and Solbakk Svein Arne "Integration of Heterogeneous Systems", Norwegian text, SI publication no. 87 762 - 2, Oslo, Norway, 1987, ISBN 82-7267-952-3.

[10] C. J. Date "An Introduction to Database Systems" Third Edition, © 1981 Addison-Wesley Publihing Company, Inc.

[11] Bruce Jay Nelson "Remote Procedure Call", Dr. Thesis, May 81, Xerox Palo Alto Research Center, report no. CSL-81-9, Palo Alto, CA, USA.

[12] Gro Oftedal "The use of Remote Applications from a Smalltalk Workstation", Master Thesis, Jan. 87, Centre for Industrial Research (SI), Oslo, Norway.

[13] K.R. Dittrich, A.M. Kotz, J.A. Mulle "A Multilevel Approach to Design Database Systems and its Basic Mechanisms", Proceedings IEEE COMPINT, Montreal 1985.

(14) D.S. Batory and A. Buchmann "Molecular objects, abstract datatypes, and data models: A framework", Proceedings of VLDB, pp. 172-184, 1984.

[15] P.P. Chen "The Entity-Relationship Model: Towards a Unified View of Data", ACM Transactions On Database Systems, 1:1, pp. 9-36, 1976.

[16] G. Di Battista and C. Batini "Design of Statistical Databases: A Methodology for The Conceptual Step", Information Systems, Vol. 13, No. 4, pp. 407-422, 1988.

[17] Trygve Reenskaug "A Methodology for Design and Description of Complex Object-Oriented Systems", Version 0.1, SI publication no. 87 01 26 - 1, Oslo, Norway, 1988 , © 1988 Centre for Industrial Research, ISBN 82-411-0103-1.

[18] O. Lehrmann Madsen and B. Møller-Pedersen "What Object-Oriented Programming May Be- and What It Does Not Have to Be", Proceedings ECOOP'88, pp.1-20, Oslo,Norway, Eds. S. Gjessing and K. Nygaard, © Springer Verlag Berlin Heidelberg 1988, ISBN 3-540-50053-7.

[19] " Specification of Information Bases and Interfaces" Ed. Bjørn Skjellaug, SI publication no. 89 01 06 - 1, Oslo, Norway, 1988, ISBN 82-411-0129-5.

---

1 ISO - International Standards Organization
CCITT - Comite Consultatif International Telegraphique et Telephonique.

2 Computer Integrated Manufacturing.

3 Distributed International Manufacturing Using existing and developing public Networks. Partly supported by the Commission of the European Communities (DGXIII-F), NTNF - The Royal Norwegian Council for Scientific and Industrial Research - (grant no. IT.3.31.22917) and The Finnish Government's Technology Development Centre (project no. 4200/87), and partly by the project members.

4 Open Document Architecture. Standard by ISO and CCITT

5 Electronic Data Interchange For Administration, commerce and trade. Standard by ISO and CCITT

6 Defence Advanced Reseach Project Agency - US Department of Defence

7 Initial Graphics Exchange Specification, published by U.S. Department of Commerce. NBSIR 88-3813.