DDTK Project: Analysis, Design and Implementation of an ODP Application

Spiros ARSENIS[a,b], Noëmie SIMONI[a], Philippe VIRIEUX[b]

[a] TELECOM Paris, Dept Réseaux, 46 rue Barrault, 75634 Paris Cedex 13, France. e-mail: arsenis@res.enst.fr
[b]DOXA INFORMATIQUE, 41-43 rue des Chantiers,78000 Versailles, France. e-mail: philippe.virieux@doxa.fr

In this paper, recently concluded work on the development of a communication support tool kit is reported. The analysis, design and implementation steps of this project follow closely the Open Distributed Processing reference model (RM-ODP). We have used the ODP concepts and design rules as the basis of our development. In this paper we present the results of ODP's application on a specific case along with all the additional elements resulting from our case analysis.

Keyword Codes : C.2.4.; D.2.10.; H.4.3.
Keywords: Distributed Applications, Object Orientation, Modelling

1. Introduction

In this paper we present the research activity carried on at both TELECOM Paris and Doxa Informatique for the realization of the Doxa Distribution Tool Kit (DDTK). Our research has been done within the framework of a pilot project for a telecom operator. DDTK provides communication support for distributed applications. Using portable communication tools as well as data presentation tools based on standard application interfaces, DDTK supports applications within distributed and heterogeneous environments. It administrates these tools in order to manage the communication from one application to an other and reuse the called treatments. The global objectives are:

- the provision of a client/server communication model supporting message exchanging through message queues,
- the treatment of the heterogeneity of a distributed environment as well as the interoperability between the different sub-sets constituting the information system,
- the securization of the system by considering the information distribution, the heterogeneity as well as the decentralisation of responsibilities,
- The administration of the transport network, the telecommunications infrastructure as well as of applications.

To cope with the complexity of distributed systems and to meet the applications requirements in terms of distribution, interworking, interoperability and portability we have followed the overall framework of ODP that provides us with design rules and guidance to meet the previously mentioned requirements. Following the five ODP viewpoints, we pass from the analysis, to the design and implementation of DDTK.

In this paper, considering the complexity of the subject, we focus our presentation on the development work concerning the DDTK communication module and its basic mechanism: the message queue. Following the five viewpoints' presentation order, we will present the modelling results of DDTK communication module development using ODP concepts as well as all the additional concepts resulting from our case study. The different viewpoints language models, basis of our analysis and design, are detailed. Starting with the requirements analysis of the enterprise viewpoint, we will arrive to DDTK implementation in a certain hardware and software context.

2. Enterprise Viewpoint

The enterprise viewpoint should serve as the basis for specifying system or application goals on which all other viewpoint specifications will directly or indirectly depend. It explains and justifies the application functions by describing its global objectives as well as its integration in the information system concerning organisation aspects and in terms of member roles, actions, purposes, usage and administration policies.

The analysis of the enterprise viewpoint is recursively applicable to several levels: from the system level considering the set of system applications, to a specific application or a specific application component. For our case study, we begin with the tool kit analysis to focus progressively to the message queue communication model analysis, the basis of DDTK.

To estimate DDTK functional characteristics we analyse the enterprise information processing requirements. This analysis is guided by the following questions :

- who requires DDTK information processing?
- which activities and which purposes are concerned with DDTK information processing ?
- which are the different roles obligations (contracts) in a DDTK community ?

To respond to the first question, let us consider the different roles in the system :

- the *user* of DDTK requiring a service meeting his needs,
- the *system designer* who wants the system to be robust and easy to maintain, analyse and expand,
- the *system manager* whose objective is to maintain normal condition of the system;
- the *developer* of the applications that installs or/and develops the applications, tests them and makes them operational in the specific context of a system.

We can attempt the consideration of viewpoints inside the Enterprise viewpoint corresponding to each of the identified roles. Each of them has different functional requirements in order to reach its own objectives and considers the DDTK application from a different point of view.

There is no proper user's perception for DDTK in terms of human being. As users of DDTK, we consider any Application Process (AP) that at a given time requires DDTK service. The APs perception of DDTK service is the provision of a client/server communication model supporting message exchanges through message queues. For these APs, distribution exists only in terms of distribution requirements. They want to communicate with each other without any preoccupation for the localization of the service, for the possible existence of heterogeneity, for the security and the supervision of the service (QoS).

For the three other roles, the distribution exists and has to be considered and provided among a number of other system properties.

The system designer considers DDTK as a brick of his system and looks for the properties that will guide his system toward openness and integration and/or any other specific policy meeting the information system needs. To the system designer pursuit, DDTK leads toward reusability of application components by breaking the applications into functional units that interact with each other through well defined public interfaces. It also provides application portability to various platforms and distribution processing across heterogeneous environments (evolutivity). At last, it provides transparency (location, relocation, access and replication transparency) and security functions.

For the system manager's pursuit for manageability and QoS, DDTK provides functions for efficient configuration, monitoring and control of the tool behaviour and resources, in order to support evaluation, prediction, QoS, accounting and security policies.

For the developer of Client/Server applications, DDTK is perceived as a tool kit offering a message queue communication model but also interface specification and compilation, trading, data conversion, and transparency functions as well as installation and testing functions in order to facilitate applications conception, realisation and operation.

To summarise and also to answer the second question (DDTK, what for ?), concerning the roles we have the following *viewpoints of responsibility*:

- The AP that requires distribution. It will define its global requirements, usually in terms of service, distribution and transparency needs. Its viewpoint is limited to the DDTK basic function: communication between applications through a message queue mechanism and research of services provided by others applications (trading),
- The system designer's viewpoint, who sees in DDTK the service offered to the users but mostly the appropriate properties that cope with his system policy,
- The administrator's and developer's viewpoints, who have to satisfy the user's requirements and use DDTK tools in order to manage all the distribution problems and provide the required service (message queue communication model).

After having analysed the required basic function that DDTK has to provide to the user, as well as the functions provided to the other roles, we can proceed to the presentation of DDTK community contract in order to answer to the last question.

The set of applications using DDTK in a distributed system form one or more communities. DDTK introduces an additional object into the system: the message queue. It is an agent object providing APs service access to the system services as well as the management of this access. Normally, there are two basic actions performed for a AP : service request and service provision corresponding to a service request. The APs (DDTK users) maintain the objectives they had before joining the community but, in the DDTK context, they are always requesting a service search, sending or receiving messages. They post and retrieve messages in and from the message queue.

The DDTK vision of the system is through an organisation structure of services materialised by the queues. To cope with the different organisation structures of enterprises (topological, infrastructure, thematic structures, etc.), a trading mechanism is introduced that permits service selection and localization depending on the characteristics asked by the users and the characteristics of the available services. The administrator determines which trading information can be accessed by whom.

Thus, an AP service requester, after selecting the appropriate queue, posts to it a message containing his service request. The request will be retrieved from the message queue by the AP service provider, will be treated and the eventual result (response) will be placed into the queue waiting for the requester to retrieve it. The kind of action an AP can perform for his communication with another AP is fixed before the beginning of message exchange.

3. Information Viewpoint

The main purpose of the information viewpoint is to specify the entities that model the system and their relationships. We will pass from an abstract explicit definition of our application (*enterprise viewpoint*) to an abstract but well structured definition (*information viewpoint*), concerning the structure, the flows, the values and their interpretation of the application information as well as the constraints of time and coherence.

To pass from our application definition to its information modelling, a methodology must be provided. The basis of this methodology is an abstract, object oriented and recursive information model (figure 1) that helps us obtain a common basis for understanding the general behaviour of the application as well as of the system in which it is installed [1] [2]. It can model every component of the distributed system and can represent both system resources and services and also identify logical partitioning, composition and inference over information.

Using the *abstraction concept* we describe any system as a composition of interacting objects. Abstraction is a fundamental way of treating complexity. Abstraction is means of ignoring the minor differences between elements and focusing on their similarities. Thus, a node-link representation can be applied to any distributed system and its applications (e.g. telecommunication service, accounting application, inventory control) considered as a set of application objects (nodes) and relations (links) that support their communications. A link is a communication object with two access points as defined in [Inoue 90].

We represent the model in figure 1 by using a simple object notation where a rectangle represents an object class: the first part contains the name of the class and the second its attributes.

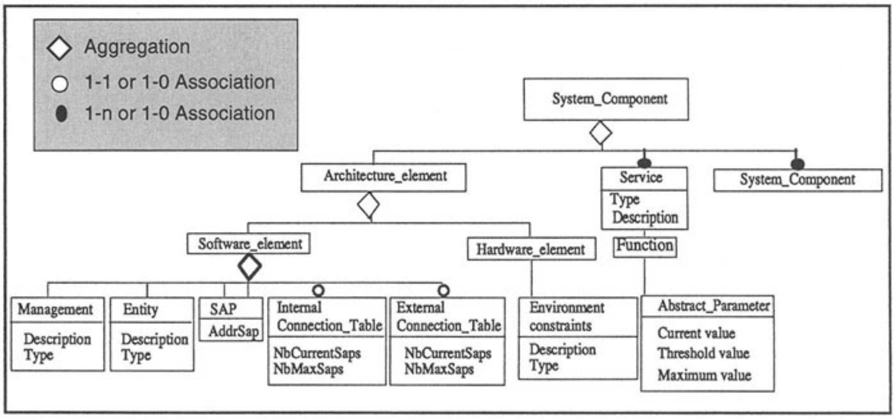

Figure 1 : Abstract information model

A *System Component* is a generic object. It may represent a domain, a node, a certain application, an application mechanism, a link, etc. Described according to the object concept, it is characterised by its own *architecture*, the *service* it provides (the system activity in which it participates) and the *SCs* it contains (*encapsulation*). Indeed, a SC carries the information of the SCs it contains since it depends on them in order to operate. Decomposition is applied recursively until all aspects of the information have been specified. Instantiation for a specific SC is realised through inheritance.

In terms of *architecture*, a SC is modelled as a composition of a software and a hardware part.

The software entity contains all the static, invariant and dynamic information of this SC. The entity class contains all the information describing the activity basic function (e.g., financial broker trading function). As the administration of any SC is one of our basic preoccupations, it has to be considered from the design phase. Thus, the *Management* contains all the necessary information (counters values, state values, etc.) permitting us to monitor and control the SC behaviour in order to reflect changes in the system but also to retrieve qualitative information on its behaviour.

The Service Access Point class, SAP, (to be compared to the Termination Point defined in [ITU92]) gives information on how users may access the entity and the Connection tables express the relations with other SCs. More precisely, the *external Connection Table* contains the access points of the (external) SCs that can be reached by this SC. The *internal Connection Table* contains the associated access points of the embodied objects.

The *service* part is modelled as a set of applicative and management functions, i.e., financial broker trading functions, fault-tolerance function, performance management function, etc. Every function is modelled as a set of abstract *parameters,* each one having current, threshold and conception values. As an example, the focus on fault tolerance may exhibit the availability, reliability, error rate and loss rate parameters.

The hardware entity contains all the information concerning the implementation or installation of this SC in a particular community (with a specific application policy), a particular station and operating system using a specific programming language (environmental constraints). It describes all the environmental constraints to be satisfied during the design and implementation steps (engineering and technology viewpoints).

The information model provides us with the image of the system concerning DDTK activity (figure 2). In a physical node one or more DDTK kernels are installed. Each one contains a number of messages queues associated with a number of Application Processes (AP) having the role of services providers. Other local (in the same Kernel) or distant APs establish accesses with the message queues in order to obtain the offered services.

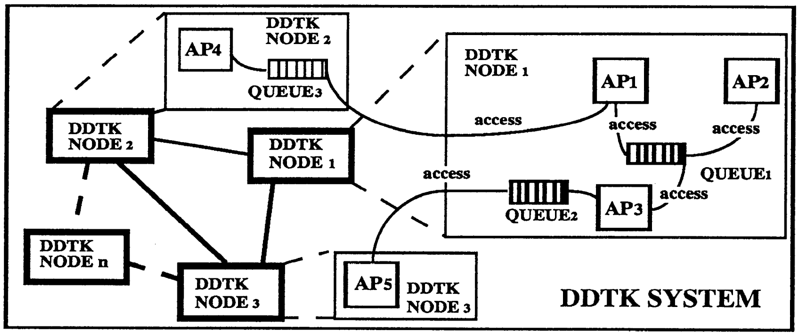

Figure 2 : Image of the information system concerning DDTK activity

Figure 2 represents DDTK application distributed over a system. Figure 3 illustrates the DDTK application using our model. The service in which this SC participates is DDTK activity: route the information through message queues connected by accesses. An access is conform to the link definition. Using the *decomposition concept* recursively enable us to specify all the aspects of the information concerning the APs, queues and accesses, that this SC contains as well as their relations.

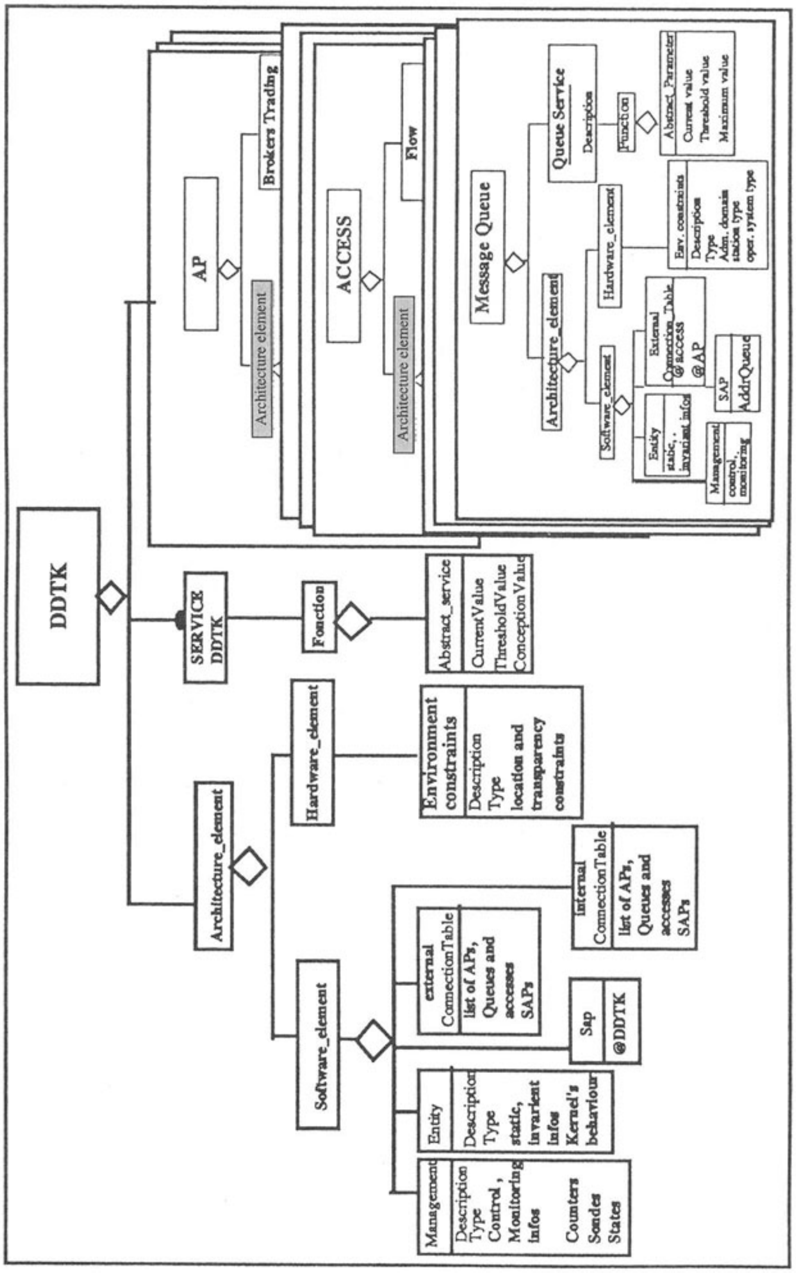

Figure 3 : Instantiation of the information model for DDTK application

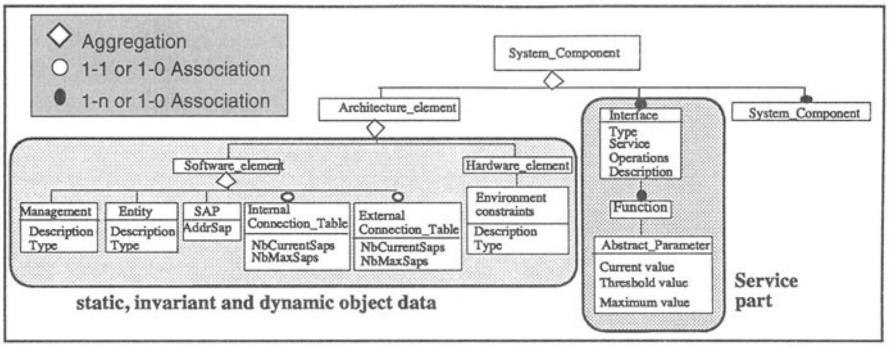

Figure 4 : Computational model

The information specification is completed with the use of state diagrams describing the states of the SCs concerning their usage, operation and administration as well as the rules that lead to the transitions from one state to an other (e.g., integrity rules imposed by the system policy). The model provides us information on the information partitioning within different administration domains. Using the diagrams, we can define the integrity rules permitting the co-operation of various sub-systems each of them having a different policy.

4. Computational viewpoint

The computation viewpoint focuses on the functional decomposition of system activities into computation interacting objects providing application specific functions. The basic aim of this viewpoint is to define the application objects configuration through a computational model describing the computational objects, their relations and specifying their interfaces as well as to define the object interaction model.

4.1. Computational model

We consider DDTK application as a set of cooperating objects that interact with each other through interfaces. Every object encapsulates internal states (dynamic object part), data and other objects (static part). It can modify their internal state by realizing treatments on their internal data. These treatments, resulting from the object interaction, are invoked by operations which are grouped into interfaces.

Let us consider once more the information model (figure 1). The architecture element contains all the static, invariant and dynamic information describing the object. The service part contains the services in which the object participates. These services correspond to a set of applicative and a set of management treatments (functions). For the computation viewpoint purposes, we express these services through the interfaces that are provided or requested (Figure 4).

We consider that an object has 3 types of Operational Interfaces: Service Provider interface (SP), through which it receives operations and provides the requested services; Service User interface, through which it requests services (SU); and the administration interface (ADM) for the administration purposes. Instantiation for a specific object is realized through inheritance, e.g., an agent has the three types of interfaces and can have more than one SP and SU interface types (he can have the client and server role at the same time for different interactions), an artefact has the SP and ADM interface types, etc.).

The functional decomposition of the abstract activity objects (information viewpoint) to the functional objects (computational viewpoint) is guided by the activity analysis results (in terms of roles, objectives, policies, etc.) of the enterprise viewpoint. These results permit us to identify the basic functions of each activity as well as its distribution requirements, each of them translatable in the objects operations and satisfied by a distinguished object function. Therefore, the user's role requirements of the enterprise viewpoint are reflected throught the SP interface, the administrator role throught the ADM interface and so on. With the computational model (figure 4) we capture all the information on the computation specification. As we notice, the model does not change from one viewpoint to the other. Its service part is refined in order to describe how the objects, that participate on specific services, interact with each other through interfaces of specific types (SP, SU) using operations selected for an operational interface signature. It also specifies which internal function (treatment) each operation invokes and describes it in terms of specific function parameters.

4.2. Interaction model

DDTK supports the applications break down into client/server modules. It proposes a C/S communication model based on message exchanges through message queues.

The Application Processes (APs) that use DDTK can be clients or servers or both. They have access to the DDTK service through an Application Programming Interface (API) that provides communication and trading functions. The APs interact using DDTK service primitives (operations) that we call messages. The primitive signature consist of an operation name and the number, names and types of argument parameters as well as the results of the primitive execution. The primitives convey to the message queue the information (messages) destinated to the server or to the client.

There are two types of AP operation mode : *synchronous mode*, where the AP is blocked waiting for the response of the execution results and *asynchronous*, where the AP does not wait for the results.

The message queue object is always collocated with the server. For the client APs the queue object provides the access to a system service (a server AP) with specific characteristics concerning the service type, the service context, the provided QoS but also specifies the type of interaction between the two objects in terms of synchronisation, QoS, etc.

A binding object supports the binding between the client APs and the queue and satisfies their environment contracts in terms of subtyping relationships between the environment contracts. The object binding is initiated by the APs using specific service primitives and specifying the queue name (id). As result they receive a binding identifier. It is also their responsibility to terminate the binding.

An AP (client or server) interacts with a queue by invoking operations at the queue interfaces. The DDTK interaction model supports two types of *interrogation* for an AP toward a queue : The posting of an AP message to the queue through a binding object and the retrieval of an AP message from the queue through a binding object. The clients post service requests and retrieve service replies. The servers retrieve services requests and post the replies.

Two queue types exist:

- *bi-directional* queues, support the emission of a client's request to the queue, the reception of this request by a server, the emission of the response to the queue and it's reception by the client.
- *Unidirectional* queues, support only the emission and reception request.

As we can see the selection of a queue "types" also the "dialogue" between the client and the server AP, imposing a type scheme that defines the APs interaction and assures type conformance. A client by choosing a bi-directional queue, chooses a synchronous "dialogue" mode as the queue guarantees, among other the presence of the server. The uni-directional queue supports an asynchronous "dialogue" that does not require the simultaneous server's presence. Moreover, the messages queues present other options as: messages save (on disk or on memory, that determines throughput performances and fault tolerance). The message queue manages simultaneous invocations of many clients to many servers. It takes care of access concurrency and also masks from the clients the presence of many servers. It transforms a simultaneous execution of operations to a sequential one.

The APs are also bound also to a trading function that permit them to select and localize a message queue by specifying the desired service characteristics as well as the queue characteristics mentioned above, and also to register a queue to the trader.

The modes of failure during interactions can be classified as binding failures (when the environment contracts cannot be satisfied, security failures (when the clients do not have access permission to a specific queue), communication failure and resource failure (due to a lack of resources).

5. Engineering viewpoint

The engineering viewpoint specifies the mechanisms supporting the distribution of the DDTK application in conformance to the computation viewpoint. It deals with the information, memory and communication proceedings necessary to support the application distribution in the context of transparency requirements, by defining their structure, their configuration, and their fine-grained placement onto the nodes.

5.1. Distribution infrastructure

In the computational viewpoint we have defined the functional objects of our application, their interfaces as well as the interaction model. The transition to the engineering viewpoint is showed in figure 5.

In this figure, we consider the interaction between two APs located in different nodes (stations). We consider a one to one correspondence between our computational objects and our basic engineering objects.

The DDTK cluster in fact represents the linking of the client or server AP with DDTK (DDTK API). The cluster configuration consists of the server APs providing a specific service (which does not forbid them from being clients of other services as well) and the message queues supporting the distribution of this service. If the client AP of the example also has the server role we should consider it as being clustered with a specific queue. The servers always have a direct binding with the message queue(s) that serves them. For the evolution of DDTK application toward servers and message queues location independence, this direct binding is also considered as a DDTK access (a link with no distribution needs for this time). The capsule represents the configuration of all the clusters and, in fact, of all the services supported by the DDTK application of a certain version. It corresponds to a unique address space. We consider these capsules to the nodes that represent the stations forming a single unit in the system in terms of location in space and embodies a set of processing, storage and communication functions.

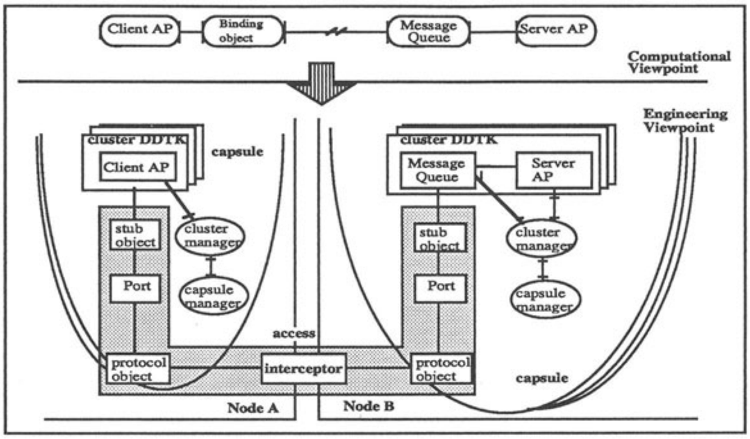

Figure 5 : Transition to the engineering viewpoint. Distribution infrastructure.

The DDTK accesses (links) correspond to the ODP channels. It is a set of objects that provide distribution mechanisms and transparency to the basic engineering objects. The access is a configuration of stub objects, binding objects and protocol objects. The stub object provides data conversion functions and supports access transparency between the engineering objects located in different stations with different operating systems, and written in different programming languages. The ports are binding objects that maintains the binding between the basic engineering objects, manage the end-to-end integrity of the access as well as the QoS of this access. They provide location transparency to the basic engineering objects. The protocol object provides the communication functions. When the protocol objects of an access are of different type they require an interceptor object to communicate.

Concerning the management functions we consider the cluster and capsule managers as part of the system global administration. We will only be interested in the management functions provided by DDTK. For the client or server AP no specific action is provided by DDTK in terms of deletion and management. The APs are clients of DDTK service, therefore is out of DDTK scope. DDTK is limited to the management of their communication. It applies controls and keep records on the AP only for security and accounting reasons (number of transactions, etc.). The queue message presents an interface with the cluster manager supporting the queue management function. Each DDTK cluster supports interactions leading to cluster creation, deletion, reactivation, disactivation, recovery and migration. It can be deleted after the desactivation or deletion of all the APs and queues it contains. The cluster management function is constrained by the management policy of it's cluster (of the APs activity and queue types). The capsule management function policy is guided by the activities that DDTK supports and mostly by the presence of DDTK communication and distribution support.

The AP, by interacting with the nucleus through the node management interface, can request an access creation that establishes a binding between the AP object and a message queue object. It can also request a message queue creation that creates the queue and generates an interface reference, to enable binding of other objects (clients) to the queue.

5.2. Engineering model

One of the basic aspects of the engineering viewpoint is the organization aspect. The engineering viewpoint imposes a structure of the basic engineering objects guided by ODP structure rules. By defining the node, capsule and cluster configuration, we specify the system organization policy. For the moment ODP basic engineering preoccupation, concerning this structure, is resource management and federation, but it can also support the others management aspects. The engineering structuring provides us with basic information on the application organization in terms of the programming language (cluster), process type (capsule), operating system (nucleus) and physical node (node). Using this information, we can organise the application operating and management in terms of activity policies in a first level and furthermore in terms of administration domain policies.

Figure 6 represents the instantiation of our model for the system SC. The system is composed of nodes that communicate with each other through links. Each node is structured as a set of capsules. The DDTK application is considered as a capsule of the node. To tell the difference between the break down concerning objects that communicate (nodes, APs, message queues, etc.), represented on the right side of the model, and the structure decomposition (capsule, cluster, etc.), we represent this decomposition on the left side of our model. The capsules are decomposed in clusters that contain the basic engineering objects (as we have already mentioned the grouping together of APs into clusters is made by activity). The basic engineering objects description (APs, accesses, stub, binding and protocol objects) is obtained using the computational model.

The engineering model contains all the necessary information for the objects and accesses realisation by considering the different environment and policy constraints of the system.

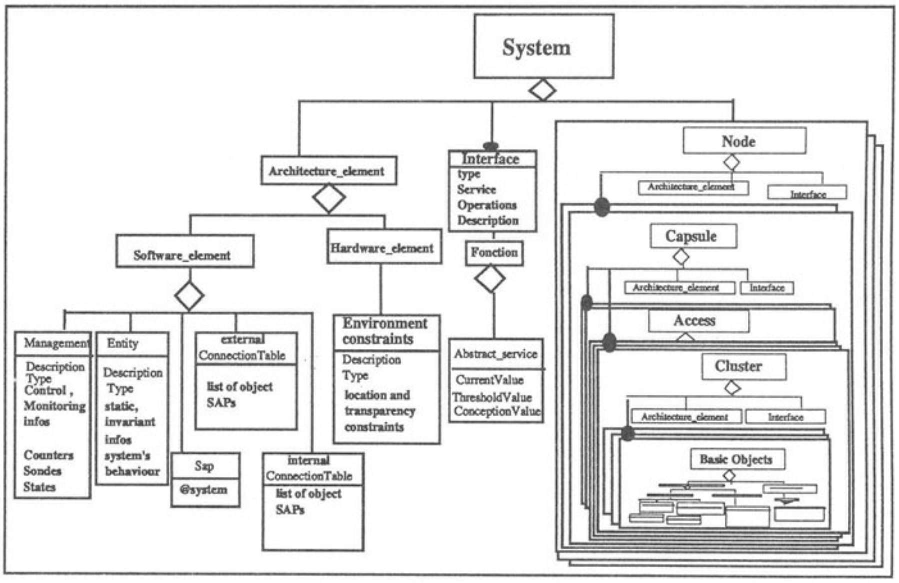

Figure 6 : Engineering model

6. Technology viewpoint

The technology specification defines the choice of software and hardware technology for the implementation of an ODP system. The figure 7 presents DDTK functions (thick line rectangular) as well as the environment that supports them.

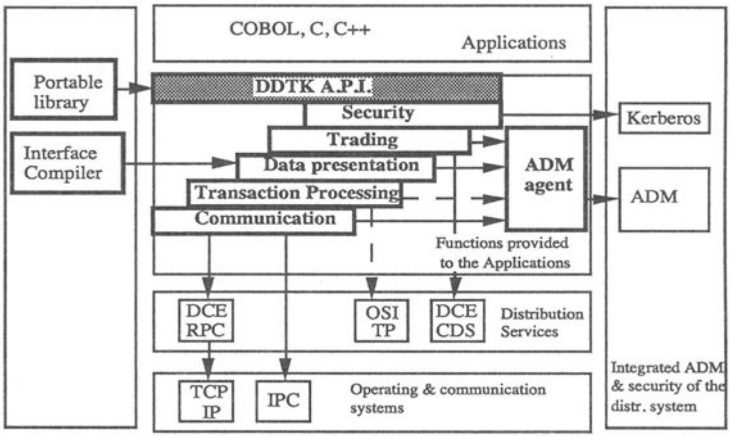

Figure 7 : DDTK functions and environment

Distributed Computing Environment (DCE) provides the distributed environment for DDTK. Although it does not fully conform to RM-ODP [5], DDTK uses it in order to build a support environment satisfying the ODP requirements.

Hence, DDTK uses: OSF/DCE Remote Procedure Call (RPC) for communication and IPC for local calls, Cell Directory Service for localization and Kerberos for security.

An Interface Definition Language (IDL) is used for describing the Application Service Interface (ISA) that links the APs with DDTK by describing the used data types and the permitted operations. The IDL is a extended version of IDL CORBA [OMG91]. It supports the construction of COBOL ISAs. A specific tool is provided, permitting the definition and description of interfaces.

The transaction processing function based on Open Distributed Systems - Transaction Processing (OSI TP) is not yet available.

DDTK is available for the moment on OS/2, AIX (BOS/X), Unix SCO and Solaris 2.3. The next stage is the porting to other environments, particularly workstation environments (Windows 3) and mainframe environments (OSF/DCE).

7. Conclusion

In this paper we have presented the development steps of an ODP application from analysis to design and implementation. ODP framework has served as the basic support providing us with basic distributed concepts, design rules and development guidance.

Following this approach, DDTK has been considered as an application developed on the ODP framework. Nevertheless, from DDTK user's viewpoint it can be considered as a middleware integrating communication and distribution tools and offering a platform solution.

8. References

[1] J. Sclavos, N. Simoni, S. Znaty. "Information Model: From Abstraction to Application", IEEE NOMS'94.
[2] S.Arsenis, N. Perdigues, N.Simoni. "Distributed Applications and Networks Integration : from modelling to implementation" TINA Fev.95.
[3] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. "Object Oriented Modeling and Design". Prentice Hall, 1991.
[4] OMG. "The Common Object Request Broker: Architecture and Specification". Dec 93.
[5] Joint Xopen/NM Forum. Translation of GDMO/ASN.1 Specifications into CORBA-IDL. Draft 0.02, July 94.
[6] A.D.Beitz, P.W.King, K.A.Raymond. "Is DCE a Support Environment for ODP?". ICODP 93
[7] Y.Inoue, M.Hoshi, H.Uchinuma, Y.Hoshi : "Transport Network Architecture for the Information Modelling" TINA 90
[8] ISO/IEC JTC1/SC 21/WG7. "Draft Recommendation X901, Basic Reference Model of Open Distributed Processing - Part1: Overview and Guide to Use", June 93.
[9] ISO/IEC JTC1/SC 21/WG7. "Draft Recommendation X901, Basic Reference Model of Open Distributed Processing - Part2: Descriptive Model", June 93.
[10] ISO/IEC JTC1/SC 21/WG7. "Draft Recommendation X901, Basic Reference Model of Open Distributed Processing - Part3: Prescriptive Model", June 93.
[11] DOXA Informatique : "General Presentation of Doxa Distributed Tool Kit" June 1994