Distributed Shared Memory Infrastructure for Virtual Enterprise in Building and Construction[1]

Fadi Sandakly[*], João Garcia[**], Paulo Ferreira[**] and Patrice Poyet[*]

[*]CSTB, BP 209, 06904 Sophia-Antipolis, France.

sandakly@cstb.fr, poyet@cstb.fr

[**]INESC, Rua Alves Redol 9-6, 1000 Lisboa, Portugal.

joao.c.garcia@inesc.pt, paulo.ferreira@inesc.pt

Abstract: This paper proposes a new approach to building a Virtual Enterprise (VE) software infrastructure that offers persistence, concurrent access, coherence and security on a distributed datastore based on the distributed shared-memory paradigm. The platform presented, Persistent Distributed Store (PerDiS), is demonstrated with test applications that show its suitability and adequate performance for the building and construction domain. In particular, the successful adaptation of SDAI to PerDiS and a comparison with CORBA are presented as examples of the potential of the distributed shared-memory paradigm for the VE environment.

1 INTRODUCTION

The software infrastructure is still one of the major difficulties for concurrent engineering development especially in large-scale projects where participants are geographically dispersed and belong to different organisations. To build such an infrastructure, traditional approaches based on remote object invocation like Corba, Microsoft DCOM and Java RMI present different limitations when applications manipulate big amounts of data. Among these limitations,

[1] This work has been partially financed by the EU PerDiS project (Esprit 22533).

three major ones should be highlighted: *(i)* performance, *(ii)* security and *(iii)* difficulties in porting legacy applications.

In fact, remote object invocation platforms on a Wide Area Network (WAN) or Local Area Network (LAN) penalise data sharing between the organisations of a Virtual Enterprise (VE). These software platforms also lack a homogenous security model defined on data and reflecting the complex security relations between partners of a VE. Finally, most of the time, porting existing legacy applications to co-operate in a VE implies their complete re-engineering in order to get decent performance, to make them work concurrently and to integrate security constraints.

This paper proposes a new approach to building a VE software infrastructure (Sandakly 1999). This approach generated a platform of persistent, distributed and shared memory called PerDiS[2]. In PerDiS, memory is shared between all applications, even located at different sites or running at different times. This shared memory represents the shared store of a VE. Coherent caching of data improves performance and availability. It ensures that applications have a consistent view of data, and frees developers from manually managing objects' location. Using the shared memory paradigm facilitates application porting on top of PerDiS. There is no need to change data structures to make them persistent and/or distributed. To allow concurrent access to data, the PerDiS platform provides transactions and locking mechanisms for connected environments and a Check-out/Check-in mechanism for disconnected ones. Locking can be handled transparently by PerDiS or explicitly by the applications.

For security, PerDiS implements a Task/Role model to manage security attributes and access rights to data. Communication between distant machines can be secured by encrypting messages. Security refers to data and users independently from applications.

[2] The source code of the PerDiS platform is freely available at http://www.perdis.esprit.ec.org/

The next section outlines the problems of developing software infrastructure for collaborative engineering in the context of VE and how PerDiS solves these problems. After that, the major concepts and implementation aspects of the PerDiS platform are detailed. This is followed by a section describing how the Standard Data Access Interface (SDAI) of the STEP ISO-10303 norm (ISO 10303-22, 1996) for Product Data Representation and Exchange was adapted to meet the requirements of a VE in the Building & Construction (B&C) sector. Finally, in the *Experiments* section a performance comparison between PerDiS and the well-known Corba approach is presented.

2 COOPERATIVE ENGINEERING AND VIRTUAL ENTERPRISE

Today, the quickly changing global open market pushes companies to react increasingly quickly and to adapt and modify their products. To achieve this goal, suppliers as well as contractors from different companies have to be tightly involved in the design and the production cycles. Co-operative or Concurrent Engineering (CE) (ISO 10303-22, 1996) (Wilbur, 1994) techniques have been generalised giving birth to a new form of *Collaborative Work* applied at company level instead of at personal level. The term *Virtual Enterprise* (VE) (Camarinha-Matos, 1999) refers to this kind of companies consortium. A VE can be defined as *a temporary alliance of independent organisations that come together to quickly exploit a product manufacturing opportunity.* These organisations have to develop a working environment to manage all or part of their different resources toward the attainment of their common goal. Obviously, common information definition and sharing is the essential problem of the VE (Hardwick, 1996). Actually, partners of a VE usually have different business rules and information infrastructures. Being part of a VE means that a company has to adapt its information system (or part of it) to the VE common information infrastructure in order to share and exchange project data with other partners.

The main issues of such infrastructure are:

1. *scalability*,
2. *evolution*,
3. *ease of use and adaptation*.

Regarding scalability, the architecture of a VE infrastructure has to be independent of the number of partners, the projects they work on, and the type and amount of data they manipulate. Furthermore, due to the fast evolution of Information Technologies (IT), this architecture has to be open and easy to evolve. At the same time it has to be simple to use in order to reduce the cost of adapting the IT infrastructure of each partner to it. The main challenges of developing this kind of infrastructure are:

- Definition of a *common data model* representing the information to be exchanged and shared between partners,
- Definition of a *common sharing software infrastructure* that can ensure data storage, data integrity, and data security.
- Adapting the existing IT infrastructure of each partner in the VE to work with the *common data model* and the *common sharing software infrastructure*.

2.1 VE in Building & Construction

The B&C sector is intrinsically *distributed* (Kalay, 1998). Generally, participants on a construction project belong to several independent companies. More than 80% of these companies are SMEs (with an average of 20 employees). Due to their size, these companies are not able to invest much in modifying their IT infrastructure each time they work on a new project. Several aspects characterise the co-operative engineering in VE in the B&C sector:

1. *Short term co-operation implies simple and fast setup of computer infrastructure for data sharing*: Except for very big projects and some special actors like the owner of a building, companies operate for a limited time in a VE specially during the design phase. For instance, architects, structure engineers, HVAC (Heating, Ventilation and Air Conditioning)

engineers and electrical engineers work for a short period on a project just to deliver the plans to be used by the construction companies. These actors stay involved in the project until the end but they become less active than others do. This relatively short-term Cupertino aspect constrains the companies to setup and adapt their computer infrastructure quickly to be able to share data with the other members of a given VE. Sharing data in a VE means being able to pick up data coming from other partners in a variety of formats and being able to exploit them without semantic loss.

2. *Long transactions in the conception cycle and the permanent need of data availability*: The conception process is cyclic. Starting from a given version of the project plans, designers add new elements and modify existing ones. Modifications made in parallel may be conflicting. Usually, a reconciliation phase between actors allows them to validate a new version of the plans and to restart a new conception cycle. Conception cycles are relatively long (in the order of weeks) and a user working session on a project can be relatively long (few hours). This working mode stimulates the notion of disconnected work session. There is no need to keep computer remote connection when the user is working with local copy of the data for a few days. Check-out/check-in mechanisms seem well adapted to the remote data access. Furthermore, attention must be paid to data availability. When a user is working on a part of a project, other users can't modify it. However, they must be able to access the *latest valid version* of this part without being blocked by another user’s transaction. Another issue raised by long transactions is fault tolerance: users cannot accept to loose a couple of hours of work because of a remote machine's crash, a network breakdown or simply because of inappropriate locking of remote data.
3. *Large datasets*: Another important characteristic of B&C design applications is the large amount of data in a project (just think about the number of different objects that exist in a

building like walls, windows, doors, stairs, electrical components, sanitary installations…).

4. *Data security, ownership and responsibility*: The legal responsibility of data is an important issue in VE mainly for engineering aspects related to the human security (fire, material resistance…). Sharing data across open networks (like Internet) must take into account the authentication of received data and its security against modifications. Only authorised persons may access and change pieces of data. Beside the legal responsibility, financial aspects increase the weight of security issues. In fact, the VE model encourages the development of a unique data store for a whole project including financial information. Protecting this data is an essential concern for all partners.

2.2 Traditional Approaches to VE software

During the last few years, an important effort has been undertaken in different research projects to define the software infrastructure of the future VE. Among the significant research efforts in this domain is the NIIIP project (NIIIP, 1996) that aims at developing open industry software protocols that allow manufacturers and their suppliers to effectively interoperate as if they were part of the same enterprise. NIIIP bases its architecture on emerging standards (formal and *de facto*) like STEP ISO-10303 (Fowler, 1995) for data modelling and Corba (OMG, 1997)(Mowbray, 1996) as a middleware for interoperation of different applications. Influenced by the OMG and Corba formalisms, NIIIP views VE activity as a set of Services offered by partners with interfaces defined with the IDL language and based on Corba standard services (OMG, 1998) like Naming, Persistence, Security, Transaction, etc.

At European level, an equivalent effort has been realised with the VEGA Esprit project that aims to establish an information infrastructure to support the technical and business operations of Virtual or Extended Enterprises. This information infrastructure relies on the Corba Access to STep models (COAST) architecture (Koethe, 1997), which allows applications to access data using an API that extends the Standard Data Access Interface (SDAI) (ISO 10303-22, 1996) of STEP. Like NIIIP, the VEGA COAST platform provides services that can be used by VE partners such as the Conversion service allowing the mapping between different data schemas and the Workflow service to manage projects. Another effort has been undertaken at the University of Salford (Koethe, 1997). In this work, a three-tier architecture is defined based on Corba services and ObjectStore OODBS for data persistence. The ISO STEP modelling language is used to define the data model. Applications can exchange data using files or can share fine grain objects whose interfaces are defined in IDL.

While ISO STEP seems to offer a modelling language and methodology, as well as data models that are largely accepted and used in different manufacturing sectors (Aerospace, Building & Construction, Electronics…), the Corba approach presents, from a practical point of view, some difficulties to building VE software infrastructure. Those difficulties can be summarised as follows:

- Currently, few of the commercial object request brokers (ORB) implement all services needed for a real VE like security, transaction, concurrency control and persistence.
- Corba is based on *remote method invocation*. With this approach, objects used by an application reside in a remote host and can only be accessed via their functional interface. This is a disadvantage when objects are frequently accessed (which is the case in most design tools, e.g. CAD) because an important amount of processing time is wasted in communications. This increases non-useful network traffic considerably.
- Most of the Corba services like concurrent access, persistence and security are defined at the level of objects. This means that important problems like concurrent data access, security and data distribution (which is a major issue in the application performance because of the remote method invocation mechanism) have to be solved in early phases of application's design.

- VE infrastructure has to integrate *existing applications* from different partners. Those applications have to access common data stores and thus have to be interfaced using Corba mechanisms. These require a deep modification in the application's data structure (the inheritance graph has to be changed to access some Corba services) and sometimes a new code structure in order to provide their functionality as a *service*.

3 THE PerDiS APPROACH

The PerDiS platform has been developed to overcome many limitations of traditional approaches in term of performance, security mechanisms, distribution capabilities and ease of use. The use of the *distributed shared-memory* paradigm as the base for PerDiS implementation improves the performance comparison to remote invocation approach notably. Furthermore, it facilitates porting of existing applications without major modifications. Additionally, transactions can be transparent to applications. PerDiS offers a default behaviour where *locking data* is done implicitly depending on application access mode to each *datum* on the store. This implicit behaviour simplifies the extension of a single-user application to a *concurrent application* where several users can share the same data since PerDiS guarantees data integrity with its locking and transactional mechanisms.

In comparison to other object distribution approaches based on remote invocation like Corba, Microsoft DCOM and Java RMI, which impose small grain distribution, PerDiS allows applications to choose their own distribution granularity based on *clusters* (which is a set of objects of variable size). As said before, B&C applications, like CAD systems, manipulate big amounts of data. Applications using remote call to access to each object attribute spend most of the execution time in network communications. With PerDiS the whole object (more precisely, a bunch of objects) is transferred once from the remote site and mapped into the memory of the local application. The network cost is widely reduced because all object accesses are done in the application memory.

Regarding security, PerDiS defines security attributes on the data store making them independent from different applications. All these aspects will be detailed in the rest of this section to show how they offer coherent data sharing, persistence and security and how they reduce time and programming effort when porting applications on top of PerDiS.

3.1 PerDiS Architecture

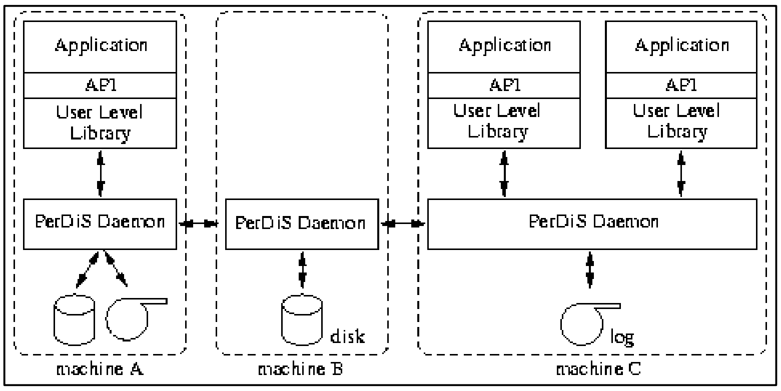


Figure 1: PerDiS Architecture

A PerDiS system (Shapiro, 1997) consists of a set of machines running two kinds of processes: application processes and PerDiS daemons (PD). There is a PerDiS daemon on each machine. Applications communicate with their local PD through a user level library (ULL) which they access via the PerDiS API. The ULL deals with application-level memory mapping, data transformations and management of objects, locks and transactions. When the application, through the API and the ULL, requests locks or accesses data, it makes requests to the local PD, which deals with caching, issuing locks and data, storage, transactions, security and communication with remote machines. A typical configuration is shown in Figure 1. Note that application processes are optional. In fact, the PerDiS architecture is a symmetrical client-server architecture in which each application is a client of the local server and all servers interact in a peer-to-peer mode. A machine with just a PD running behaves as a pure server.

3.2 *Objects and clusters*

Objects in PerDiS are sequences of bytes representing some data structure. They are not limited to e.g. C++ (Stroustrup, 1986) objects. An application programmer allocates objects in a cluster, which is a physical grouping of logically related objects. Clusters have a variable (unlimited) size. In contrast with current technology like Corba (OMG, 1997), clusters are the user-visible unit of naming, storage, distribution and security, allowing efficient and large-scale data sharing applications. In fact, a cluster combines the properties of a heap (programs allocate data in it) and of a file (it has a name and attributes, and its contents are persistent). Programmers use URLs (Berners-Lee, 1994) to refer to clusters, e.g. pds://perdis.esprit.ec.org/clusters/floor1.

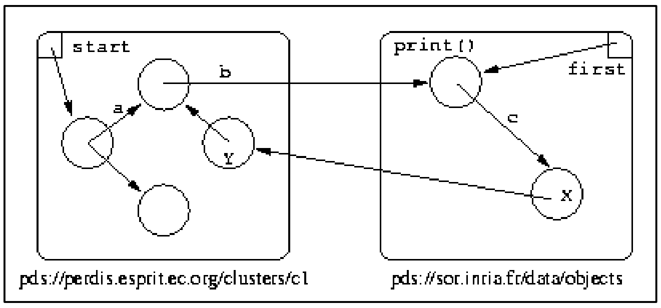

start
b
a
Y
print()
first
c
X
pds://perdis.esprit.ec.org/clusters/c1
pds://sor.inria.fr/data/objects

Figure 2: Clusters, objects and remote references

An object in some cluster may refer, using standard pointers, to some other object in the same cluster or in another one, even when the current machine does not actually hold the pointed-to object. While navigating through an object structure, an application may implicitly access a previously "unknown" cluster, which may be on a distant machine. For instance, Figure 2 shows two clusters located at two different machines. Starting from the entry point *start* in the leftmost cluster, one can navigate through the objects e.g. `start->a->b->print()`, implicitly accessing the remote cluster. The same function could have been called by first opening the remote cluster and then calling `first->print()`.

3.3 Persistence

Persistence in PerDiS is based on persistence by reachability (PBR) (Atkinson, 1983): an object is persistent if and only if it is transitively reachable from a persistent root. A persistent root is a distinguishable, named reference, which is persistent by default. To illustrate PBR, reconsider Figure 2: all objects are reachable from the roots *start* and *first*, and thus they are persistent. If the pointer `first->c` is set to `NULL`, objects X and Y would no longer be reachable, and would automatically be deleted. Destroying the root objects *start* and *first* would delete all objects in both clusters.

The PBR model has two main advantages: it makes persistence transparent and frees programmers from memory management. Persistence is transparent since it is deduced from the reachability property. It frees the programmer from memory management because programmers only allocate memory; deallocation is performed by the system if necessary. This prevents dangling pointers and memory leakage. When porting existing code, only the allocation operator has to be modified to allocate data in persistent memory. There is no need to modify or extend data structures to make them persistent.

PBR is implemented with a Garbage Collector (GC). The GC runs as a thread of the PD. It scans the store regularly to eliminate unreachable parts of the persistent store.

3.4 Caching and replication

Data distribution in PerDiS is based on lazy replication and co-operative caching. Lazy replication means that the system only makes copies of data when an application accesses them. Co-operative caching means that caches of different PDs interact to fetch the data an application accesses. Replication and caching avoid the remote access bottleneck because all data access is local. In addition, since replicas are kept in several co-operative caches, data requests are spread over several nodes preventing data access bottlenecks. A potential drawback of sharing implicitly sized units of data is false sharing. False sharing occurs when two applications access different data items that happen to be stored in the same unit of locking. Each application has to wait until the other has unlocked the data before it can access it, although there is no real sharing.

Another important aspect of caching implemented in PerDiS is the *prefetching mechanism*. In fact, PerDiS allows an application to specify a prefetching strategy depending on its behaviour. For instance, the application can define a set of clusters that the cache prefetches when a given cluster is opened by the application. This mechanism enhances data availability specially when it is located on a remote machine with a slow network connection.

3.5 Transactions

The purpose of transactions is to guarantee fault tolerance and concurrency control to PerDiS applications.

PerDiS provides transactions with the usual ACID semantics (Atomicity, Consistency, Isolation, and Durability) while supporting both optimistic and pessimistic concurrency controls. Pessimistic concurrency control enforces locking when data is accessed. Locked data is unlocked when the transaction is committed. Optimistic concurrency control allows users to access data concurrently without locking. Conflicts between optimistic transactions are detected through data versioning at commit time. Data is marked with a version stamp every time it is written.

Furthermore, the PerDiS model allows for non-serializable views of data (private copies), for notifications and for reconciliation transactions. The broad range of these transactional facilities is motivated by the application domain of co-operative engineering. Interactive project development applications (e.g. CAD) may issue long transactions which are unlikely to involve write conflicts but which, due to their length and complexity, users will want to avoid aborting at all costs.

Transactions using implicit locking automatically lock data accessed by the application. Data read by an application is protected by a read lock, whereas data that is to be modified is protected by a write lock. Modifying data that is already protected by a read lock causes an upgrade of the lock from *read* to *write*. Implicit locking is mainly provided to make porting existing applications to PerDiS easy. However, it reduces concurrency, because it is done at memory page granularity, and increases the risk of deadlocks. Therefore, when developing new applications, explicit locking is preferred. Explicit locking transactions do not use automatic locking, but rely on explicit object intent requests, using functions like `lock()`, and `unlock()`.

When starting a transaction, a programmer may indicate that the application must be notified when the involved data is accessed, modified or committed by other transactions. This allows the development of reactive applications. Applications can, for instance, react to these types of events by updating their cached data using a "refresh" function and initiating a new transaction in order to reconcile with concurrent users.

All these functionalities are further extended in PerDiS through its two-level architecture. In PerDiS, there is a distinction between local area (LAN) and wide area (WAN) networks. For each LAN, there is a node responsible for interacting with PerDiS servers on remote networks. This site, called a gateway, includes a file cache and provides any files from machines outside the LAN to local area PerDiS applications. Each PerDiS server manages a multi-version store where a sequence of versions for each file is kept as they are submitted for commit by PerDiS applications. Executing transactions on a WAN using PerDiS does not require that all servers involved are synchronised. PerDiS uses a transaction commit algorithm derived from MVGV (Agrawal, 1987) that synchronises only involved servers at commit time. Furthermore, this algorithm allows the file cache to give coherent views of data to applications without having any knowledge of transactions. The PerDiS transactional protocol extends notifications and reconciliation functionalities to wide area transactions.

3.6 Security

Protecting data in VEs is important for two reasons. First, project data often represents a large asset due to the amount of work needed to create it. Losing data means losing money. Second, data often represents knowledge and provides companies with a competitive edge. Due to the nature of a VE, partners co-operating in one project may be competing in another. Therefore, PerDiS provides security means (Coulouris, 1997) to protect data in a collaborative environment. Security in PerDiS consists of two parts: data access control and secure communication. Data access is controlled by groupware-oriented access rights based on users' tasks and roles. PerDiS clusters are assigned to a particular task and one can specify access rights for a user having a specific role in a specific task. Access rights can be assigned on a cluster basis to reduce management overheads. When using a secure PerDiS application, users are associated to their role in the task by logging on to a PerDiS security tool.

Secure communication uses public key schemes for signed data access requests, shared keys for encryption and a combination of the two for authentication of message originators.

4 DISTRIBUTING STEP SDAI

To open the PerDiS platform to the ISO Standard for the Exchange of Product Model Data (STEP) formalism and to allow its integration in the Building & Construction domain, the authors ported an implementation of the Standard Data Access Interface (SDAI) (ISO 10303-22, 1996) on top of PerDiS. This interface was defined for single user applications accessing a single local or remote data store. This section details how to proceed in order to extend SDAI to make it deal with data distribution, security and multi-users concurrent access to data stores. The main goal of this work is to facilitate the development of STEP concurrent applications. The size of the SDAI layer is about 50000 lines of C++ code. This code has been ported to PerDiS in a very short time comparing to the effort of its development.

The rest of this section describes STEP and SDAI shortly and is followed by a discussion of the issues involved in developing a distributed SDAI application using PerDiS.

4.1 The International Standard for the Exchange of Product Model Data

STEP provides a basis for communicating product information at all stages of a product's life cycle. The keyword of data exchange in STEP is *Data Model Sharing*. In fact, STEP defines tools like the EXPRESS language (Atkinson, 1983) to develop data models that can be used in different applications allowing interoperability and common data structures for data sharing. EXPRESS is an OO-like language providing mechanisms to model constraints on data like *Global Rules* and the *Uniqueness* of object values. STEP provides also a definition for data exchange (ISO 10303-21, 1994), which is an ASCII format that can be used to exchange data defined with EXPRESS. This file-based format is called STEP Physical File Format (SPF). Finally, for data storage and access, STEP specifies an application programming interface (API) called SDAI (ISO 10303-22, 1996) (STEP Data Access Interface) that defines the way applications can store and retrieve instances in databases. The goal is that applications sharing the same data model can share databases. However, SDAI does not deal with *concurrent access* of data or with *distribution* of databases. Data security is not defined in the SDAI either.

4.2 Architecture of the SDAI

The SDAI is defined in four different schemas written in EXPRESS (see Figure 3):

- The SDAI **Dictionary Schema**: It includes definitions of entities needed to represent a meta-model of an EXPRESS schema. Instances of these entities that correspond to a given schema constitute the SDAI Data Dictionary.

- The SDAI **Session Schema**: It includes the definition of entities needed to store the current state of a SDAI *session* started by an application. The information stored is mainly the list of repositories opened by the application and their access mode, current transactions, events and errors.
- The SDAI **Population Schema**: It defines the organisation structure of a SDAI *population*. A SDAI population is the set of instances stored in SDAI repositories. Three main entities to store instances are defined in this schema (see Figure 4):
    1. *SDAI Model*: is a grouping mechanism consisting of a set of related entity instances based upon one schema,
    2. *Schema Instance*: is a logical collection of SDAI Models based upon one schema. A schema instance is used as a domain for EXPRESS *Global Rules* validation, as a domain over which references between entity instances (in different models) are supported or as a domain for EXPRESS *Uniqueness* validation.
    3. *Entity Extent*: it groups all instances of an EXPRESS entity data type that exist in a SDAI Model.
- The SDAI **Parameter Data Schema**: This schema describes in abstract terms the various types of instances that are passed and manipulated through the API. It provides definitions for EXPRESS *simple types*, EXPRESS *entity instance*, EXPRESS entity *attribute value*, *aggregations* and *iterators*, etc.

Those schemas are independent of any implementation language. The SDAI language bindings (SDAI Implementations) are specified for computing languages like C, C++, Java and IDL.

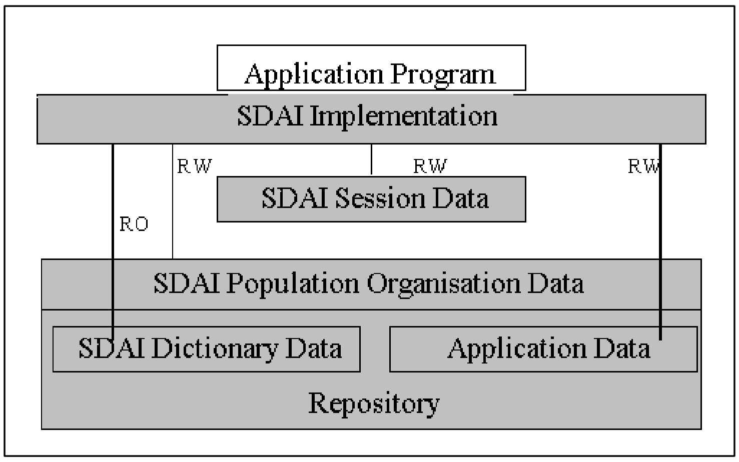


Figure 3: SDAI Architecture

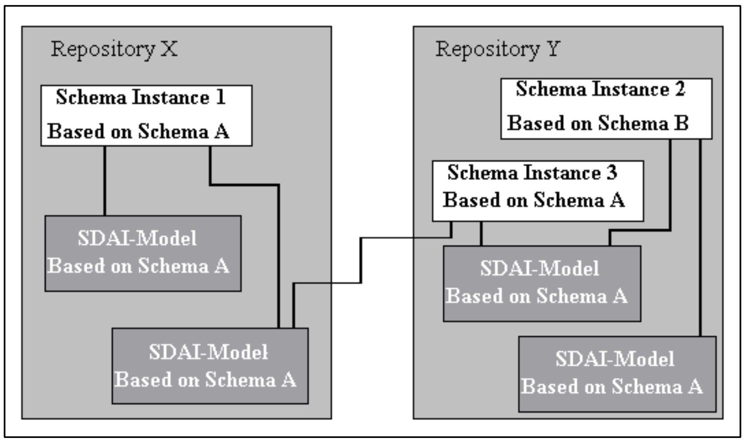


Figure 4: SDAI Storage Structure

4.3 Porting SDAI on the PerDiS Platform

This section discusses the main issues in developing a persistent distributed version of a SDAI, which allows concurrent data access. Some of the design decisions made have been influenced by one of PerDiS’ principal goals which is the fast porting of existing applications that were neither developed to work in a concurrent environment nor in a transactional way. To do this the PerDiS team preserved as far as it was possible the SDAI API described in (ISO 10303-23, 1997) by introducing *implicit* concurrent and transactional behaviour based on some of PerDiS features. However, SDAI was extended with a specific distribution and concurrency API that can be used by new applications.

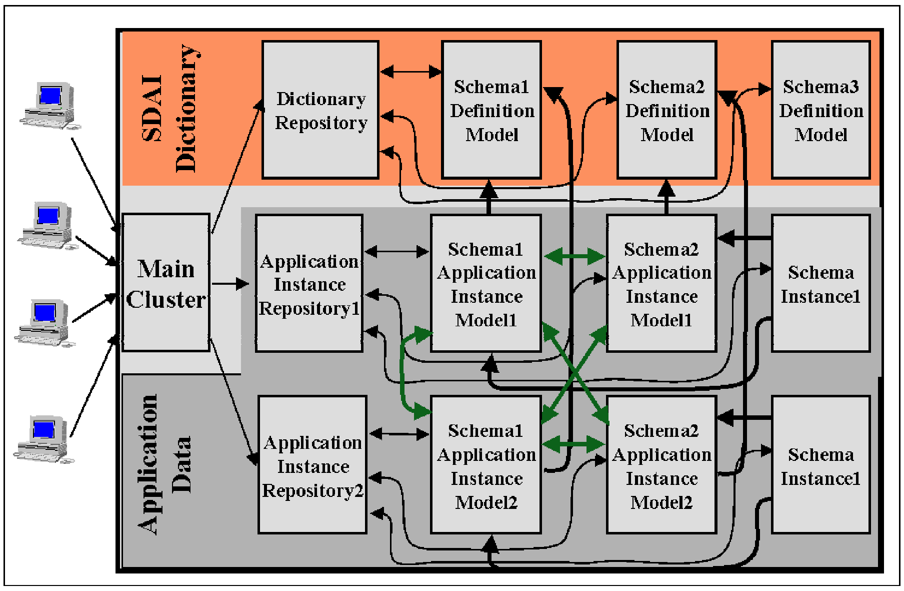
4.3.1 SDAI Persistence and Storage Structure

All instances created in the SDAI are made persistent in PerDiS. This is simplified by the fact that in PerDiS no difference exists between persistent and transient object manipulation.

In a SDAI implementation, application instances are stored in a logical structure. These structures are placed in the PerDiS persistent distributed store (*i.e.* in clusters). The main object *collector* in the SDAI is the *Model*. From the application point of view, a model is a set of related instances. Another important collector is the *Repository*. A repository is a collection of models. Those two objects are part of the application data organisation and have to be persistent. *Schema Instances* are also logical collectors that contain several models related to the same schema. As shown in Figure 5, all these objects are mapped to PerDiS clusters that are a logical collection of application objects.

4.3.2 Distribution Granularity

In the PerDiS approach, the distribution granularity as seen by the application is a cluster. Physical distribution is hidden and the PerDiS platform can manage this granularity automatically or semi-automatically to optimise performance using prefetching mechanisms (see section 3.4). From the SDAI point of view, the smallest collector structure is the model. Implementing SDAI models with clusters gives applications fine-grain distribution granularity. This granularity is adjustable because models can contain any number and kind of application instances. Inside a cluster, instances can be grouped using entity extent objects.



SDAI Dictionary
Dictionary Repository
Schema1 Definition Model
Schema2 Definition Model
Schema3 Definition Model
Main Cluster
Application Instance Repository1
Schema1 Application Instance Model1
Schema2 Application Instance Model1
Schema Instance1
Application Data
Application Instance Repository2
Schema1 Application Instance Model2
Schema2 Application Instance Model2
Schema Instance1

Figure 5: Persistent distributed SDAI implementation in PerDiS. Boxes represent PerDiS clusters

4.3.3 Concurrent data access

There is no concurrent access specification in the current definition of the SDAI. As shown in Figure 5, the SDAI data are divided into two categories: Dictionary data and Application data. Normally, the Dictionary data is read by all applications and it is not modified often. The default access to the dictionary is implemented with the *local-copy* locking mode. This mean that the application accesses the last valid version, but modifications to this data are not committed except when the applications create a new EXPRESS Schema (which means adding a new data model to the dictionary). In this case, the locking mode is upgraded to *read-write* and the application can be blocked if another one modifies the dictionary at the same time.

Application data category represents the project data that users modify most frequently. Usually, the project data is organised in different SDAI models. This organisation reflects the project decomposition in term of tasks and users. For instance, architecture elements (like walls, openings…) can be stored in several SDAI models. Each one of these models may correspond to a part of the building (*i.e.* a floor) under the responsibility of an architect (*i.e.* a user). Each user needs to access its own data (its part of the project) to modify it. On the other hand, he may occasionally need to access some other parts for viewing or comparison needs (the last valid version is needed). This pattern of work guided this implementation of the default concurrent access to the Application data. By default, SDAI models (i.e. PerDiS clusters) are locked in *read* mode. PerDiS automatically upgrades this mode to *read-write* when the application modifies the data. When the user needs to access parts of the project belonging to other users, applications can open SDAI models with *local-copy* mode to avoid being blocked when someone else is modifying the model. Data locked in *local-copy* mode can have its locks upgraded if applications need more control of it. This approach simplifies the porting of single-user STEP applications to make them work concurrently when they were not designed to do this.

4.3.4 Transactions

Once a SDAI session is started, an application can manipulate instances in stores in a transactional context: the application *starts* a transaction, *locks* objects (implicitly or explicitly), *accesses* its data, and finally it can *commit* the transaction or *abort* it. By default, transactions are mapped to PerDiS *pessimistic transactions*. The standard API is extended to support the *optimistic* transactions when applications do not want to block others and/or are able to perform reconciliation when conflicts arise.

4.3.5 Security

The security model in PerDiS is defined at two levels:

1. Security of communications which is transparent to the application,
2. Security of stored data, expressed as *access rights* given to different users in a VE according to a *task-role* model (Coulouris, 1997). This model separates the security attributes from the data model definition. It allows the management of *legal* and *data ownership* aspects in a VE. No major modifications have to be done to port applications that don't deal

with security except the handling of access rights violation that can be done at the highest level of an application.

Access rights are defined using the security managing tools developed in PerDiS. The mapping of the SDAI storage structure to the PerDiS clusters allows us to use those tools with the SDAI implementation without modifications. SDAI models become the user security units.

5 EXPERIMENTS

Besides the complete SDAI layer ported on top of PerDiS, several applications have been developed or ported to PerDiS to manage different aspects of VEs and to test the performance of the platform (like security management tools, Xfig [a drawing tool for UNIX], OO7…). This section presents two applications; the first is a mapping program that translates architectural data from the ISO AP225 (ISO 10303-225, 1996) format to the VRML (ISO 14772-1, 1997) format. This program has been also ported on top of a Corba ORB (Orbix) and its performance has been compared to PerDiS. The second application is a *document management application* that allows to manage sets of files distributed over several servers and to access them transactionally. This application shows the use of PerDiS in transactional disconnected operations with different kinds of data types.

5.1 AP225 to VRML Mapping Application

ISO AP225 (ISO 10303-225, 1996) is a standard format for representing building elements and their geometry; it is supported by a number of CAD tools. The application presented here reads this format and translates it into VRML (ISO 14772-1, 1997) (Virtual Reality Modelling Language), to allow a virtual visit to a building project through a VRML navigator. This application was chosen because it is relatively simple, yet representative of the main kernel of a CAD tool. The original, stand-alone version is compared with a Corba and a PerDiS version.

The stand-alone version has two modules (see Figure 6). The *reader module* parses a SPF (STEP Physical File Format) (ISO 10303-21, 1994) file containing a building project, and instantiates the corresponding objects in memory. The *generator module* traverses the object graph to generate a VRML view, according to object geometry (polygons) and semantics. The object graph contains a hierarchy of high-level objects representing projects, buildings, storeys and staircases. A storey contains rooms, walls, openings and floors; these are represented by low-level geometric objects such as polyloops, polygons and points.

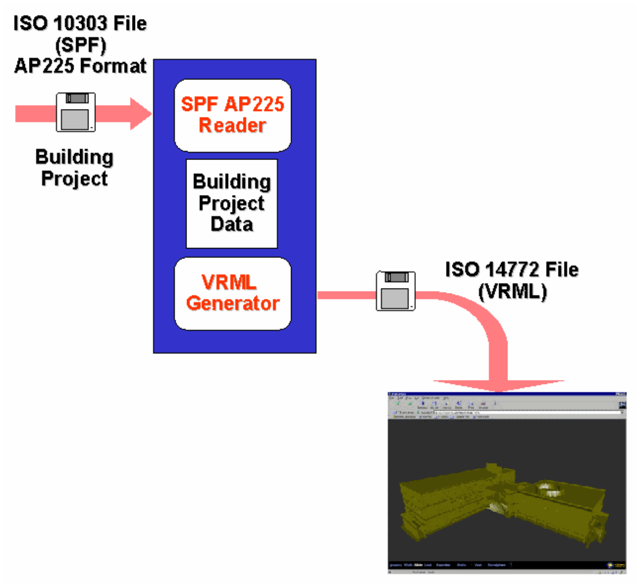


Figure 6: Stand-Alone version of the AP225 to VRML application

In the Corba port, the reader module is located in a server, which then retains the graph in memory (see Figure 7). The generator module is a client that accesses objects remotely at the server. To reduce the porting effort, only five classes were enabled for remote access: four geometric classes (Point, ListOfPoints, PolyLoop, and ListOfPolyLoops) and a class (Ap225SpfFile) that allows the client to load the SPF file and to get the list of polyloops to map. The porting task took two days (for only five classes). The code to access objects in the generator module had to be completely rewritten.

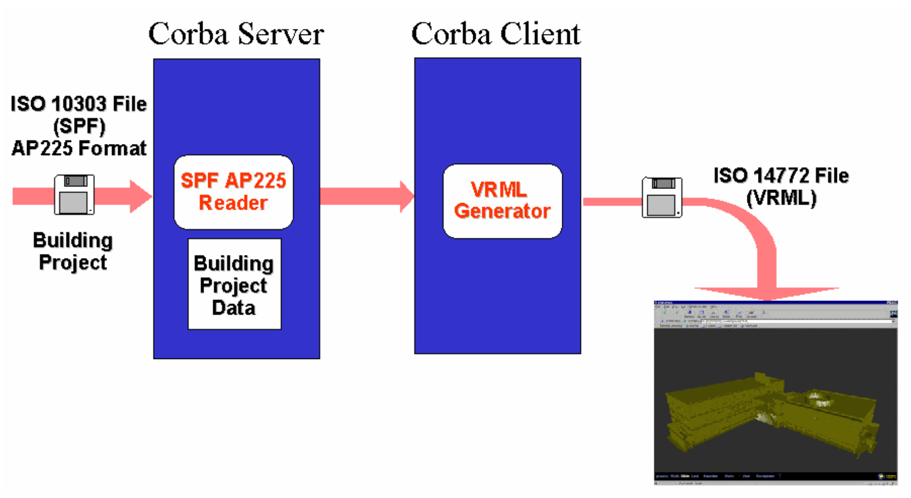


Figure 7: Corba implementation of the AP225 to VRML application

In the PerDiS port, the reader module runs as a transaction in one process and stores the graph in a cluster (see Figure 8). The generator module runs in another process and opens that cluster. The porting task took only one day and all classes were made distributed with no modification of the application architecture. The PerDiS version has the advantage that the object graph is persistent, and it is not necessary to re-parse SPF files each time. The VRML views generated are identical to the original ones.

The stand-alone version is approximately 4,000 lines of C++, in about 100 classes and 20 files. In the Corba version, only five of the classes were made remotely accessible, but 500 lines needed to be changed. In the PerDiS version, only 100 lines were changed.

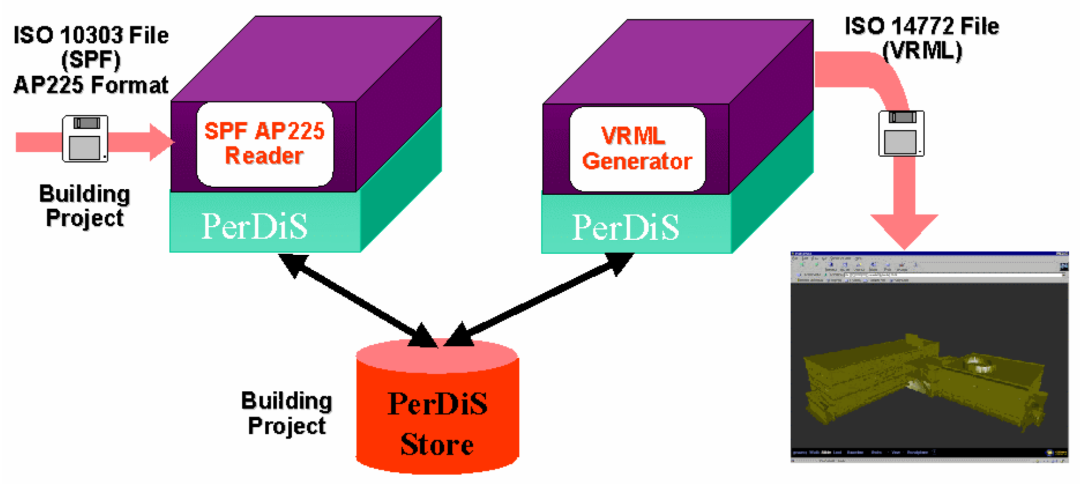


Figure 8 PerDiS implementation of the AP225 to VRML application

Table 1 compares the three versions for various test sets and two different configurations:

- Std-Alone represents the original stand-alone application.
- The **1 machine** column represents:
    1. The PerDiS version where the application accesses data stored on the same machine.
    2. The Corba version where the server and the client run on the same machine.
- The **2 machines** column represents:
    1. The PerDiS version where the application accesses data stored on a different machine.
    2. The Corba version where the server and the client run on two different machines.

Compared to a remote-object system, even a mature industrial product such as Orbix, the PerDiS approach yields much better performance.

Table 2 shows the memory consumption comparison. This consumption in the PerDiS version is almost identical to the stand-alone one, whereas the Corba version consumes an order of magnitude more memory, for reasons that were not clear. This experience confirms the intuition that the persistent distributed store paradigm performs better (in both time and space) than an industry-standard remote-invocation system, for data sets and algorithms that are typical of distributed VE design applications. It also confirms that porting existing code to PerDiS is straightforward and provides the benefits of sharing, distribution and persistence with very little effort.

SPF Files			**Execution Time (s)**				
File Size In Kb	N° of SPF Objects	N° of Polyloops	Std-Alone	**On 1 Machine**		**On 2 Machines**	
				PerDiS	Corba	PerDiS	Corba
293	5200	530	0.03	1.62	54.52	2.08	59.00
633	12080	1024	0.06	4.04	115.60	4.27	123.82
725	12930	1212	0.07	4.04	146.95	5.73	181-96
2031	40780	4091	0.16	13.90	843.94	271.50	1452.11

Table 1: Execution time comparison between the 3 implementation of the AP225-VRML mapping application: Stand-Alone version, PerDiS version (on 1 and 2 machines) and Corba version (on 1 and 2 machines)

SPF Files			**Memory Occupation (Kb)**			
File Size (Kb)	N° of SPF Objects	N° of Polyloops	Std-Alone	PerDiS		Corba
				In Memory	Persistent	
293	5200	530	2269	2073	710	26671
633	12080	1024	2874	2401	1469	51054
725	12930	1212	3087	2504	1759	59185

Table 2: Memory occupation comparison

5.2 Project Manager Application

The second application presented in this paper is a document management application called ***Project Manager***. This application aims at allowing applications not programmed for PerDiS to manage sets of files distributed among several servers and to access them transactionally. Sets of conventional files (text files, spreadsheets, CAD files...) which are related and needed as a group in order to perform an activity can be put into a common project. The project manager is coupled with a browser and any file at a PerDiS site, which is made visible by the local WWW server, can be added to a project. The coherence of the view provided over these sets of files is guaranteed by transactions performed within the Project Manager. When a project is created and files are inserted into it, these files are included in the PerDiS system and therefore from that moment on their coherence, distribution, persistence and concurrency control are ensured by PerDiS. Every time a user wants to change files belonging to a project, he (she) starts the project manager and checks out the files to a location of his choice (local directory, portable computer, and floppy disk...). The project manager can then be terminated and only has to be started again when the user wishes to check in the project's files. Mean-while, any external application can be used to view or modify the files. When the files are checked-in, a transaction is committed which guarantees the ACID properties of the sequence of changes made by the external application.

6 CONCLUSION

Today, the major difficulty to build a Virtual Enterprise consortium is the lack of a software infrastructure that can integrate persistence, distribution, concurrent access control, data integrity, coherence and security. In addition, there is a clear need for an efficient interface that allows a fast porting of existing applications to reduce the development effort needed to ensure the interoperability of different tools from different partners. This paper presented PerDiS, a new approach to develop VE software infrastructures using a persistent distributed store, which is based on lazy replication and a global co-operative coherent cache. This approach reduces the performance problem by avoiding the remote access techniques used by traditional distribution approaches, i.e. based on RPC. Furthermore, PerDiS includes features such as implicit locking and optimistic and pessimistic transactions to allow a simple and fast porting of applications that were not developed to work in a transactional and concurrent environment.

Another major difficulty of VE is the definition of common data models. For this purpose, the authors proposed the standardisation efforts as a solution. ISO STEP techniques and methodologies start to be widely accepted in different manufacturing sectors. To open the PerDiS platform to a wide range of application domains a distributed version of the Standard Data Access Interface (SDAI) to develop concurrent STEP tools was implemented. Finally, the paper presents a performance comparison of a VRML mapper ported to PerDiS and to an industrial implementation of Corba. More details about PerDiS concepts, implementation, performance and applications can be found in (Ferreira, 2000).

Acknowledgements

The authors would like to thank all partners of the PerDiS Esprit project 22533 who contributed to the design and the development of the platform and the test applications. Members of PerDiS consortium are (in alphabetical order): CSTB (France), IEZ (Germany), INESC (Portugal), INRIA (France) and QMW (Great Britain).

7 REFERENCES

Agrawal. D., Bernstein. A. J., Gupta. P., Sengupta. S. (1987) Distributed optimistic concurrency control with reduced rollback. Distributed Computing, 2, 45-59.

Atkinson. M.P., Bailey. P.J., Chisholm. K.J., Cockshott. P.W., Morrison. R. (1983) An approach to persistent programming. The Computer Journal, 26(4), 360-365.

Berners-Lee. T., Masinter. L., McCahill. M. (December 1994) Uniform Resource Locators. RFC 1738.

Camarinha-Matos. L.M., Afsarmanesh. H. (1999) The Virtual Enterprise Concept. Infrastructure for Virtual Enterprise, Kluwer Academic Publishers, Boston, USA.

Coulouris. G., Dollimore. J., Roberts. M. (1997) Security Services Design. PerDiS deliverable PDS-R-97-008. http://www.perdis.esprit.ec.org/deliverables/docs/T.D.1.1/A/T.D.1.1-A.html.

Faraj. L., Alshawi. M., Aouad. G., Child. T., Underwood. J. (1999) Distributed Object Environment: Using International Standards for Data Exchange in the Construction Industry. Computer-Aided Civil and Infrastructure Engineering, 14, Blackwell Publishers, New Jersey, USA.

Ferreira. P., Shapiro. M., Blondel. X., Fambon. O., Garcia. J., Kloosterman. S., Richer. N., Roberts. M., Sandakly. F., Coulouris. G., Dollimore. J., Guedes. P., Hagimont. D. and Krakoviak. S. (February 2000) PerDiS: Design, Implementation and Use of a PERsistent Distributed Store. Recent Advances in Distributed Systems, Krakowiak S., Shrivastava S. K., Lecture Notes in Computer Science 1752, Springer-Verlag, Heidelberg, Germany.

Fohn. S. M., Greef. A., Young. R. E., O'Grady. P. (1995) Concurrent Engineering. Lecture Notes in Computer Science, 973. Springer Verlag, Heidelberg, Germany.

Fowler. J. (1995) STEP for data management, Exchange and Sharing. Great Britain: Technology Appraisals. ISBN 1-871802-36-9.

Hardwick. M., Spooner D. L., Rondo T., Morris. K. C. (February 1996) Sharing Manufacturing Information in Virtual Enterprise. Communications of the ACM, Vol. 39, 2.

ISO 10303-11. (1994) Industrial automation system and integration-Product data representation and exchange- Part 11. Description methods: The EXPRESS language reference manual.

ISO 10303-21. (1994) Industrial automation system and integration-Product data representation and exchange- Part 21. Implementation methods: Clear Text Encoding of the Exchange Structure.

ISO 10303-22. (1996) Industrial automation system and integration-Product data representation and exchange- Part 22. Implementation methods: Standard Data Access Interface specification. 1996.

ISO 10303-225. (1996) Industrial automation system and integration-Product data representation and exchange- Part 225. Application Protocol: Building Elements Using Explicit Shape Representation.

ISO 10303-23. (January 1997) Industrial automation system and integration-Product data representation and exchange- Part 23. C++ programming language binding to the standard data access interface. ISO TC184/SC4/WG11 N004.

ISO 14772-1. (1997) The Virtual Reality Modelling Language.

Kalay. Y. E. (1998) Computational Environment to support design collaboration. Automation in Construction, 8, Elsevier Science, Amsterdam, Netherlands.

Köthe. M. (January 1997) COST Architecture: the Corba Access to STEP Information Storage - Architecture and Specification. Deliverable D301 of ESPRIT 20408 VEGA project.

Mowbray. T. J., Zahavi. R. (1996) The Essential CORBA - System Integration Using Distributed Objects, John Wiley and Sons, New York, USA.

Object Management Group. (December 1998) Corba Services Specification. http://www.omg.org/corba/sectran1.html.

Object Management Group. (September 1997) The Common Object Request Broker Architecture and Specification (CORBA) Revision 2.1. http://www.omg.org/corba/corbaiiop.htm.

Sandakly. F., Garcia. J., Ferreira. P., Poyet. P. (1999) PerDiS: An Infrastructure for Cooperative Engineering in Virtual Enterprises. Infrastructures for Virtual Enterprises, Networking

Virtual Enterprises, Camarinha-Matos. L.M., Afsarmanesh. H., Kluwer Academic Publishers, Boston, USA.

Shapiro. M. et al. (1997) The PerDiS Architecture. Deliverable PDS-R-97-002 of ESPRIT 22533 PerDiS project. http://www.perdis.esprit.ec.org/deliverables/docs/architecture.

Stroustrup. B. (1986) The C++ Programming Language, Addison-Wesley, New York, USA.

The National Industrial Information Infrastructure Protocols (NIIIP) Consortium. (1996) The NIIIP Reference Architecture (Revision 6). http://www.niiip.org/public-forum/NTR96-01/NTR96-01-HTML-PS/niplongd.html.

Wilbur. S. (June 1994) Computer Support for Co-operative Teams: Applications in Concurrent Engineering. IEEE Colloquium on Current Development in Concurrent Engineering Methodologies and Tools.