FIRST | ESA | R | 0021.10

Distributed Objects and the Prototype FIRST Integrated Network and Data Archiving System A tutorial FIRST-EMS-ARC001

JJ Mathieu (WMS), P Claes (SAF)

2^{nd} September 1998

Change Log

Who	Date	Version	Change
JJM	01JUL98	Draft 0.1	Original
PC	21JUL98	Draft 0.2	UML-section, Summary, keysteps chapter 5,6 added.
PC	10AUG98	Draft 0.3	draft-glossary included, first comments of Project(PE) included, rework of sections 5, 6 and 7
PC	28AUG98	Draft 0.4	VEGA-comments taken into account and streamlining of sections 5, 6 and 7
JJM	2SEP98	Draft 0.5	Reordering sections and including GP comments
JJM,PC	27NOV98	Draft 0.6	Reordering sections, chapter on development and development of first third party client.

Contents

1 Summary

This tutorial introduces the necessary aspects of the FINDAS-prototype to know in order to develop server and clients. It is complementary to the documents provided by the contractor which implemented its core.

Section 3 outlines the foreseen activities from December until June 1999, as well as aspects which might be worthy of further attention.

Section 4 deals with the characteristics of the prototype: capabilities, boundaries, architecture, basic components, development tools and software standards. How to use the prototype in practice (connecting, querying the database, viewing and generating the data) is explained in section 5. Software engineering aspects, standards, procedures of how to develop a new application is exemplified in section 6.

Starting with section 8 some background material is provided about Object-Oriented technology.

2 Applicable and reference documents, acronyms

2.1 Applicable Documents

FDS.PWN.001 FINDAS Concept and architecture: Technical report Issue 1 from 8^{th} June 98 by Vega Plc

FDS.PWN.002 Review of the use of CORBA in FINDAS Issue 1 from 17^{th} April 98 by Vega Plc

FDS.PWN.003 Development of a FINDAS prototype: requirements, tools and scenarios Issue 1 from 24^{th} June 98 by Vega Plc

FDS.PWN.004 Review of FINDAS user requirements Issue 1 from 26^{th} May 98 by Vega Plc

2.2 Reference Documents

Guide to applying the ESA software engineering standards to project using the Object-Oriented methods Issue 1 from March 1998

2.3 Acronyms

ACL	Access Control List
ADD	Architectural Design Document
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off The Shelf
DCL	
DDD	Detailed Design Document
DDL	Data Definition Language
DML	Data Manipulation Language
FTP	File Transfer Protocol
HTML	HyperText Markup Language
(astro) IDL	Interactive Data Language
(CORBA) IDL	Interface Definition Language
IIOP	Internet Inter ORB Protocol

OCL	Object Constraint Language
ODBMS	Object Data Base Management System
OODBMS	Object Oriented Data Base Management System
OMG	Object Management Group
OO	Object Oriented
OQL	Object Query Language
ORB	Object Request Broker
OT	Object technology
PDF	Portable Document Format
SRD	Software Requirements Document
SVVP/AT	Software Verification and Validation Plan / Acceptance Test
UML	Unified Modeling Language
URD	User Requirements Document
WWW	World Wide Web

3 Development plan

The development timeline for the prototype is as follows:

Period	Action
Until 8^{th} December	- Prepare ESA server/client for integration in prototype. - Complement in as much as possible the tutorial document.
8-11 December	- Demonstration of preliminary version of prototype. - Integration of ESA server/client.
January	- Prepare other third party client(s). - Incremental changes in current dataset client/server. - Continue elaboration of tutorial.
February	- Attendance O2 course.
March	- Installation of prototype at ESTEC. - Tutorial complemented by final documentation from Vega. - Submission of proposals for client/server from ICC's. - Evaluation of prototype and technologies.
Mid-April	- Development and integration of ICC's client/servers.
June	- End of maintenance. - Lessons learnt report.

What follows are ideas/proposals/aspects which might be explored within the period January to June. Some need the prototype, others can be explored without it:

- Connectivity between freeware CORBA and IONA's ORBIX. Choice of which freeware is important (TAO, Orbacus, ...), whether they support also Java.
- Choice of language for development and which language is best suited to which layer of the software.
- Possibly choice of Java RMI versus CORBA. The choice may not be as simple, both offer different functionalities and developers might favor "easier" environments.
- Replication aspect of O2: is it worth the extra effort versus cost, provided functionality and its current status.
- Organisation of development environment: from local software and global software, directory structure organisation, complex class structures.
- Definition of classes versus the "transport" of objects on the network (many small messages trash the communication channels).
- Guidelines of software for use within a Web centric system, what can and cannot be done (easily).

- Definition of various types of client/server (multi-threaded, server activated on method call, synchronization needed between clients, etc...).

4 Prototype

The information in this section will be complemented with data coming from the software development document and user manual from Vega.

4.1 Capabilities and boundaries

4.2 Development: general aspects

4.3 Architecture

4.4 Basic components

4.4.1 Database

4.4.2 Basic level classes

4.4.3 Workflow classes

4.4.4 Applications

4.4.5 Viewers

4.5 Tools

4.5.1 Hardware environment

Two workstations from Sun Microsystems:

- Sun Ultra 10
- 128Mb of main memory
- 9Gb and 4.5Gb disks

4.5.2 Software environment

- Solaris 2.6 and appropriate patches from Sun.
- Visual Workshop version 4.2 for C++ from Sun.
- Java Development Kit (version 1.1.x or higher).
- Sun Web server 1.0 from Sun.
- CDE Motif 1.2.6 (not yet confirmed) from Sun.
- Web browser supporting HTML 3.2 and Java (probably Netscape 3.0 or higher).
- PDF reader Acroread.

4.6 Software standards

4.7 COTS

- Rational Rose version 5 for C++ (Cost 10000 ECU for floating licence and 1 year maintenance).
- ORBIX C++ development licence with support contract and 25 runtime licences (Cost 8700 ECU)
- O_2 system comprising two sets of licences for server, OQL, O_2 C++ binding, O_2 CORBA, O_2 notification, O_2 web, O_2 replication, O_2 version (Cost 38000 ECU).

5 Using the prototype

OMost of the information if not all of this might be reaplced by the SUM provided by VEGA. Only complementary information should be put here.

5.1 Connecting to the system

5.2 Using the document repository

5.3 Data generation and viewer

5.4 Querying the system

5.4.1 Querying the database

5.4.2 Querying object servers

6 Adding new components

The description of the development follows broadly ESA standard practices and is described step for step from user requirements to completion BEFORE integration with the FINDAS prototype. It will be completed end December with the integration steps taken at Vega. Diagrams when appropriate will be substitued by their Rational Rose version.

6.1 Customer request

Develop a system, called **XYSYSTEM**, which will act as a repository for 2-dimensional data sets. Users shall be able to query the datasets which are in the repository, display selected datasets and add new datasets. Although each dataset is identified by a unique combination of creator and dataset name, they are not protected in use by other than their creator. The system will also support the user in creating and making available to the system whatever analysis tool they wish.

The system shall be implemented using ORBIX and operate on Sparc/Solaris platforms. The system shall operate within an intranet. The system shall be integrated with the FINDAS prototype. The system shall be accessible via a web browser.

6.2 Proposal

The following issues have been discussed and clarified with the customer.

1. We propose an object-oriented development framework.
2. The **XYSYSTEM** has an element of distribution, its users will be operating on a intranet, thus it is proposed to use an object request broker as the backbone of the system.
3. The visualization of curves will be performed initially by a Java applet, with the possibility to use other legacy software (possibly Gnuplot).
4. The delivered software will be minimal but expandable. The user shall be guided (constrained) by the publicly available interface definitions to create their own client software.
5. The development process for the integration of new features will be outlined and an example provided.
6. The components of the system will be defined using UML diagrams, checked by the development team and submitted for approval to the customer.
7. Deliveries will be as follows:
   (a) Local system with most functionality (including user interface), without back-end database. This first delivery shall not need to be usable via a web browser.

(b) Intranet system using a CORBA server implementing the facilities agreed on the first delivery.

(c) Replacement of the back-end file system with the prototype database facility.

(d) Integration of the front-end within a Web browser.

6.3 Requirements analysis

1. Dataset are produced by different users at undefined sites within an intranet.
2. Dataset exists beyond their creation process.
3. Data analysis tools may be created at any time and should be integrable in the system.
4. Datasets and analysis tools have a creator (a user) and a name.
5. The tuple creator and name are unique in the system.
6. A new item with the same creator and name tuple as an old dataset replaces the old one (no versions are preserved).
7. Users can retrieve items by their tuple creator and name.
8. The number of users in the system is small (say 5 users).
9. Users are registrated in the system solely with their name. There are no restriction to the registration process.
10. Access to the system uses forms which can be called up via a URL from a web browser.

6.4 XYSYSTEM use case

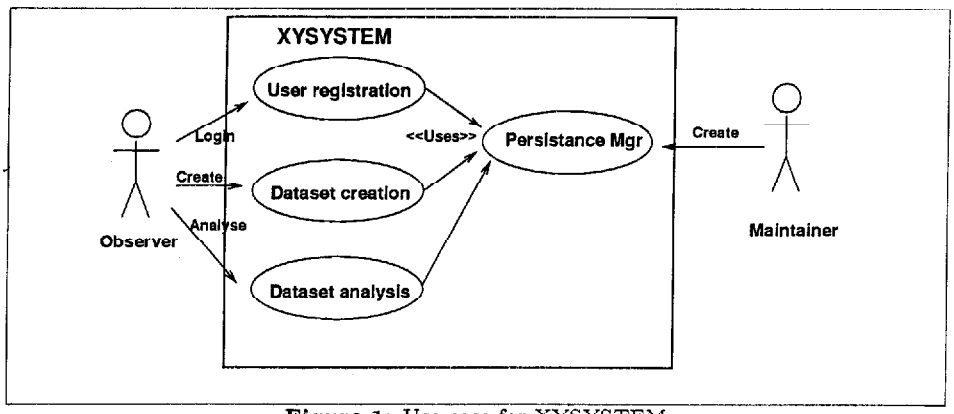

Figure 1: Use case for XYSYSTEM

6.4.1 Actors

The **Maintainer** will initialize the system by adding the various object servers to the ORB repository. The dataset and user repositories do not yet exist and will be created as a result of requests from other actors.

The **Observer** logs in to the system by calling up the home page of the XYSYSTEM which presents a login screen. As the user has supplied his name, the user repository will be eventually updated and the work screen will be shown. The working screen allows the user to display a dataset or to introduce a new data set. Furthermore the user can call up some analysis tool to operate on the selected dataset.

6.4.2 Use case description

Item	Description
Name	User registration
Description	Provide the entry point to an observer in the FINDAS system.
Actor	Observer
Initiation	Call up FINDAS home page via a Web browser fill in user name and click LOGIN button
Result	XYSYSTEM working screen is displayed and user is fully registered with FINDAS
Precondition	XYSYSTEM server is registered with FINDAS.
Postcondition	- Work sceen is displayed in browser. - User and session registered with FINDAS.

Item	Description
Name	Dataset creation
Description	Observer integrates a new dataset with XYSYSTEM.
Actor	Observer
Initiation	From the work screen the user presses the FUNCTION button.
Result	Dataset integrated in XYSYSTEM.
Precondition	- Temporary session is active. - Work sceen is displayed in browser. - Function code is in the functions directory.
Postcondition	- Dataset registered with the XYSYSTEM.
Notes	Possibility of integrating new datasets from files is also foreseen.

Item	Description
Name	Dataset Analysis
Description	Observer displays a dataset from XYSYSTEM
Actor	Observer
Initiation	From the work screen the user presses the DISPLAY button
Result	Dataset displayed in a window.
Precondition	- Work sceen is displayed in browser.
Postcondition	- XY graph display the chosen dataset.
Notes	- Only display tool provided.

6.4.3 Statechart diagram

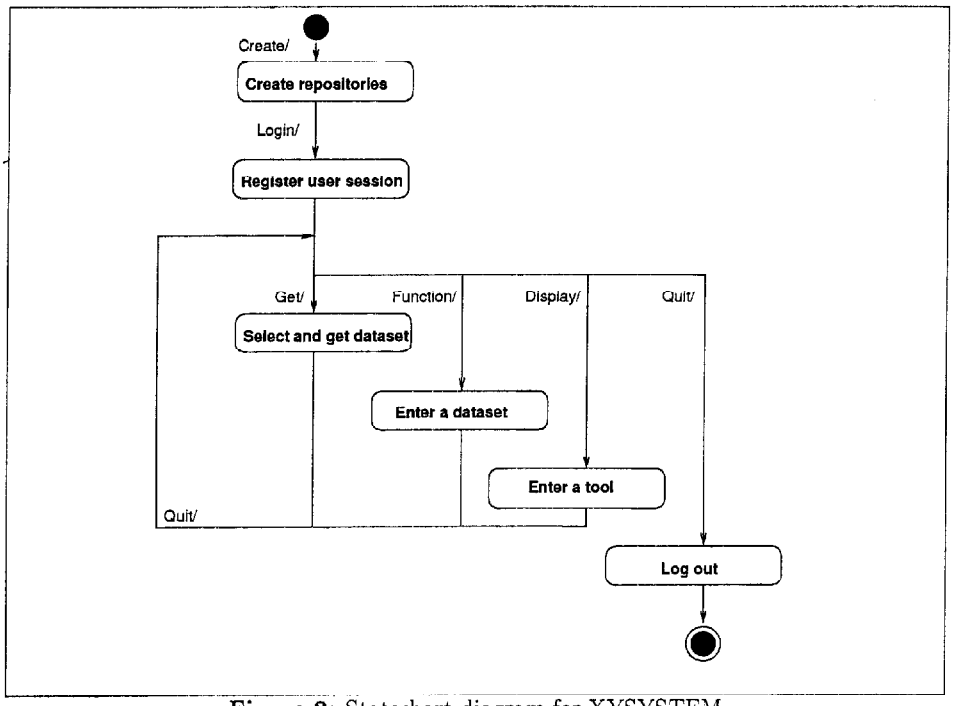

Figure 2: Statechart diagram for XYSYSTEM

6.4.4 Architecture and deployment diagrams

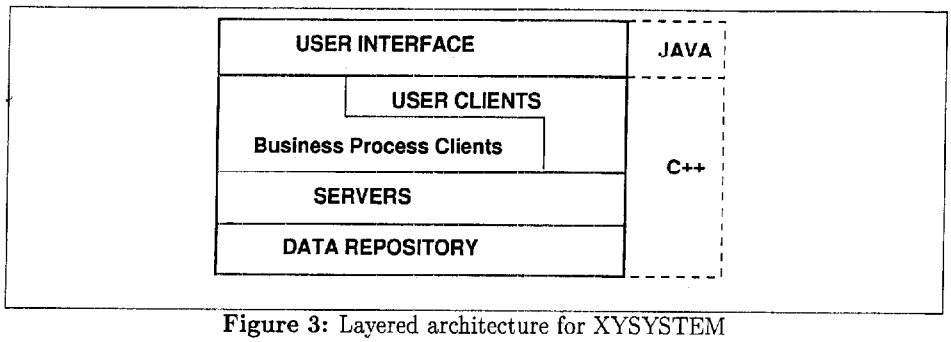

Figure 3: Layered architecture for XYSYSTEM

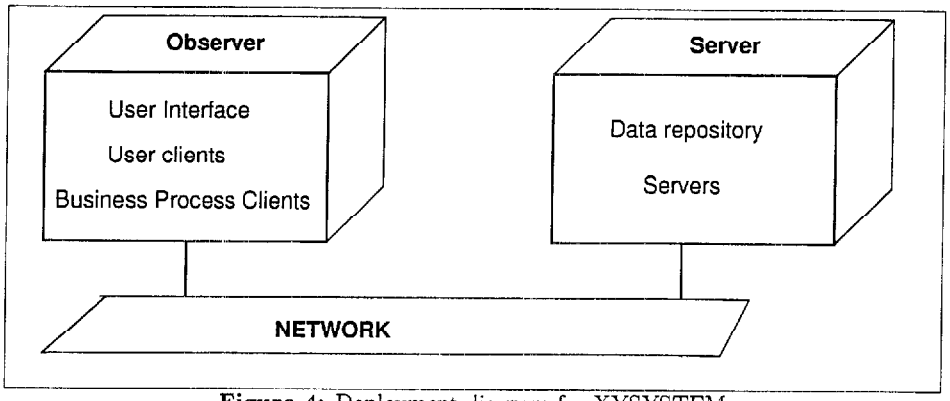

Figure 4: Deployment diagram for XYSYSTEM

In the first release of the system, both the business process and user clients are running off the Observer computer. In subsequent releases the business processes will move to the server and be part of the provided CORBA services. Also the user interface is not accessible from a browser in the first release, it relies on local file systems to preserve its current session data. The system supports simultaneous users though.

6.4.5 Class diagram

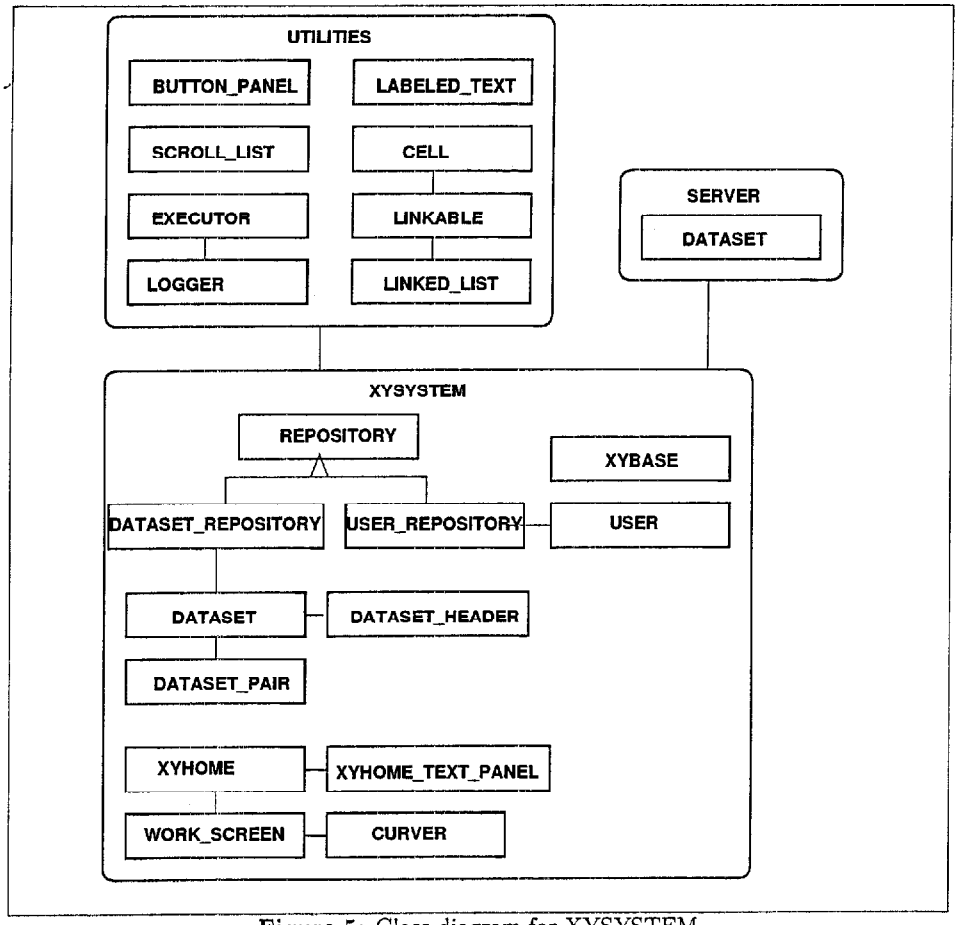

Figure 5: Class diagram for XYSYSTEM

The *UTILITIES* and *XYSYSTEM* clusters are part of the *FRONT_END*. The *SERVER* cluster forms the *BACK_END*.

CLASS	XYBASE
Purpose	Abstract class which SYSTEM classes implements.
Cluster	SYSTEM
Inherits from	
Queries	boolean Object_comparison() *virtual boolean equals(Object other)*
Commands	compare_object()
Constraints	Initialized to compare by reference.

CLASS	USER
Purpose	Holds user centric characteristics.
Cluster	SYSTEM
Inherits from	XYBASE
Queries	boolean equals(Object other) String name() String toString()
Commands	make(String name)
Constraints	Initialized to compare by value.

CLASS	REPOSITORY
Purpose	Abstract class keeping Objects in linked lists.
Cluster	SYSTEM
Inherits from	XYBASE
Queries	boolean is_initialized() boolean has(Object item) String name() boolean off() Object item()
Commands	make(String name) set_repository_name(String name) *load()* *save()* reset() add(Object item) remove(Object item) start() forth()
Constraints	Initialized to compare by value.

CLASS	USER_REPOSITORY
Purpose	Repository of known users to the system. Currently preserves the data as straight ASCII file stored locally
Cluster	SYSTEM
Inherits from	REPOSITORY
Queries	boolean has(String name)
Commands	load() save() add(String name) remove(String name)
Constraints	

CLASS	DATASET_HEADER
Purpose	Hold header information of dataset.
Cluster	SYSTEM
Inherits from	XYBASE
Queries	boolean equals(Object other) String user() String name() int count() String toString()
Commands	make(String user,String name) make(String user,String name,int count) set_user(String name) set_name(String name) set_count(int count) add_pair() remove_pair() reset()
Constraints	Initialized to compare by value.

CLASS	DATASET_REPOSITORY
Purpose	Repository of data sets, preserved as local ASCII files.
Cluster	SYSTEM
Inherits from	REPOSITORY
Queries	
Commands	load() save()
Constraints	

CLASS	DATASET
Purpose	Aggregation class for user and xy pairs.
Cluster	SYSTEM
Inherits from	XYBASE
Queries	String user() String name() int count() DATASET_PAIR item() double x() double y() boolean off() String toString() boolean equals(Object object)
Commands	make(String user,String name) nake(String user,String name,int count) set_user(String user) set_name(String name) put(double x,double y) put(DATASET_PAIR data) start() forth() reset()
Constraints	

CLASS	DATASET_PAIR
Purpose	Holds a pair of coordinates.
Cluster	SYSTEM
Inherits from	XYBASE
Queries	boolean equals(Object other) double x() double y() String toString()
Commands	make(double x, double y) set_x(double value) set_y(double value)
Constraints	Initialized to compare by value.

CLASS	XYHOME
Purpose	system user interface start class.
Cluster	SYSTEM
Inherits from	Frame
Queries	boolean accept(File directory,String file_name)
Commands	destroy() itemStateChanged(ItemEvent) actionPerformed(ActionEvent) main(String arguments[])
Constraints	

CLASS	XYHOME_TEXT_PANEL
Purpose	user interface text dialog class.
Cluster	SYSTEM
Inherits from	Panel
Queries	String host_name() String dataset_directory() String function_directory() String temporary_directory()
Commands	set_host_name(String name) set_dataset_directory(String name) set_function_directory(String name) set_temporary_directory(String name)
Constraints	

CLASS	WORK_SCREEN
Purpose	Class displays a dataset.
Cluster	SYSTEM
Inherits from	Applet
Queries	
Commands	init() destroy() itemStateChanged(ItemEvent event) actionPerformed(ActionEvent event) nake(String host,String temp_dir,String hf_name,String title,EXECUTOR executor) load_headers(String name,DATASET_REPOSITORY data)
Constraints	

CLASS	CURVER
Purpose	Draw a curve.
Cluster	SYSTEM
Inherits from	Canvas
Queries	
Commands	make(String name, int rows) paint(Graphics handle)
Constraints	

CLASS	dataset
Purpose	Interface dataset for CORBA.
Cluster	SERVER
Inherits from	
Queries	string user() string name() long count() string file_name() string internal_file_name() boolean dataset_list_off() boolean pair_off()
Commands	make(in string name,in string user,in string file_name) put_pair(in float x,in float y) pair(out float x,out float y) dataset_list_start() dataset_list_forth() pair_start() pair_forth()
Constraints	

```
// dataset.idl
// Version 1.0
// Create or show contents of a dataset
//

interface dataset {
void make(in string name,in string user,in string file_name);
void put_pair(in float x,in float y);
void pair(out float x,out float y);
void save();
string user();
string name();
```

```
long count();
string file_name();
string internal_file_name();
void dataset_list_start();
void dataset_list_forth();
boolean dataset_list_off();
void pair_start(in string name);
void pair_forth();
boolean pair_off();
};
```

6.4.6 Processes implemented

The following programs have been developed:

- *dataset* is the server which is provided to the Implementation directory from ORBIX. It is declared as an unshared and per client process activation mode. The server will be connected to the prototype's database during December.
- *xyheaders* is a business process client which lists datasets known to the server and their characteristics.
- *load_dataset* is a business process client which given a dataset name provides its contents.
- *filer* is a business process client which provides to the server a dataset given a user file.
- *xyhome.html* is the java front end program which provides access to the dataset facility. This process in the current release is not web compliant (it manipulates local files and thus is unsafe). It calls the *xyheaders*, *load_dataset*, *filer* and user defined processes to exchange datasets with the user.
- *user generators* a set of programs which interact directly with the server and provide new datasets to store.

6.4.7 Directory organisation and Makefile

The following hierarchy starting from a root directory is set:

- FILES: for user datasets to be preserved in repository.
- FUNCTIONS: directory where USER CLIENTS should be developed.
- TEMPORARY: directory used by the system to save files.
- PROGRAMS:
  - FRONT_END: contain Java front end
    - CLASSES: compiled code
    - SRC: source code (in UTIL and SYSTEM sub directories).
  - BACK_END: (CORBA) IDL and C++ for ORBIX

The PROGRAMS directory contains the Makefile with targets *all* and *clean*, which will call recursively other Makefile in the hierarchy. The BACK_END Makefile generates the server starting from the (CORBA) IDL file and will store the necessary header and object files into the FUNCTION directory, such that a rebuild of user function generation is eased.

7 Legacy components

Legacy applications are not considered within the frame of the prototype.

8 Background information

8.1 Object Technology (OT)

As mentioned in the introduction , systems were designed and operating as monolith often running on a set of closely connected computers with little networking facility (basic message support). The next step was the introduction of remote procedure calls: crudely put libraries were put on the network. The next evolution is the development of the object technology.

8.1.1 Definitions

The following definitions have been taken from the glossary of the Object Management Architecture (OMA).

- An *object* is the combination of a state and a set of methods that explicitly embodies an abstraction characterized by the behavior of relevant requests. A basic characteristic of an object is its distinct identity which is immutable and persists as long as the object exists. The OMA states also that "state data and methods can be located at one or more locations".
- A *class* is an implementation that can be instantiated to create multiple objects with the same behavior. An object is an instance of a class.
- *Types* classify objects according to a common interface while classes classify objects according to a common implementation.
- The *interface* is a description of a set of possible uses of an object.
- An *implementation* is a definition that provides the information needed to create an object, allowing the object to participate in providing an appropriate set of services.
- *Inheritance*: construction of new definition (class) by incremental modification of other definitions.

8.1.2 Object model and database

A database is a central facility which provides persistence to data (i.e. data which outlive the process that creates it). An object oriented database should ideally implement a facility which provides persistence to complete objects. As noted above the objects have both states (represented as pure data) and services (pieces of executable code). Representing data in such a way that it can be used and 'understood' by different platforms is not difficult, but storing methods which can be used on different platform is not yet possible (except possibly with Java). Thus in a object oriented system which uses a database, one needs to explode objects in two parts: the object database holds the data and sometimes also the methods signatures (i.e. the interface as defined above) and for each class an external

representation of the class which contains the code for executing the methods (i.e. the implementation part).

One can use with an object model either a pure object oriented database, or a relational database. When a relational database is used, every object has to transit via an interface which maps the table oriented database store into an object model. This process may lead to inefficiencies and eventually to mismatches of two very different data models.

Many relational database vendors propose modules which implement the mapping between the two representations. Most object oriented database systems are more recent than their relational counterpart, but the technology has matured enough to provide strong and solid products.

8.1.3 OT benefits

Major characteristics of objects technology worthwhile mentioning:

- Separation of interface and implementation: implementation can be changed as needed, providing the changes do not violate the interface definition. One should note that the interface definition only enforces the conformance of signatures (i.e. is syntactical rather than semantic). As an example the use of '+' with numbers and strings objects may in effect cover two different meanings:

$$3 + 5 = 8$$

$$"abc" + "cde" = "abccde"$$

  The result of the operation is probably not what a mathematician would expect: the additive operator in the case of strings appends one string to the other.

- Incremental development: by using the inheritance characteristic of the object model, effectively building upon already developed components.

The major benefit in using object technology comes from the fact that one can partition the system to develop in groups of related classes (usually called clusters) and develop the clusters in parallel. Doing so minimizes the complexity of solving the problem and makes the integration of the clusters easier (one does not integrate all elements in one lump, but rather proceeds by increments). The maintenance of an object system also benefits from this partitioning since one limits the number of changes to a small number of clusters.

The use of object technology does not limit the developer to only use objects. A non-object entity can be wrapped in an appropriate shell to create an object, this is referred to as *Embedding*.

8.1.4 OT and issues in system development

Thus OT will address two of the issues of software development in particular:

- Allow incremental development and integration of software.

- Support the integration of existing applications and libraries.

An issue which can be also supported by object technology is the platform independence by using the language Java. Java is a language and a also a virtual machine which interprets Java executables (byte-codes) on several different platforms.

For publications on object oriented technology and web sites please consult the appendices.

8.2 CORBA

As object technology became a main player in the field of software development, networking facilities were more and more used and thus lead to needs of combining this two aspects into a consistent and useful model. The Common Object Request Broker Architecture (CORBA) is the model enabling the inter-operation of objects on a distributed network of different computer platforms.

8.2.1 CORBA reference model

The reference model consists of four components:

- *Object Request Broker* (ORB): enables objects to transparently make and receive requests and responses in a distributed environment.
- *Object services*: a collection of services (interfaces and objects) that support basic functions for using and implementing objects. Services are necessary to construct any distributed application and are always independant of application domains.
- *Common facilities*: a collection of services that many applications may share, but which are not as fundamental as object services (for instance an electronic mail facility or a system management).
- *Application objects*: products of a single vendor on in-house development group which controls their interface. Applications objects correspond to the traditional notion of applications, so they are not standardized by the Object Management Group (OMG).

8.2.2 ORB and (CORBA) IDL

A CORBA software kit has two components: (CORBA) IDL compilers and an ORB.

The *Interface Definition Language* ((CORBA) IDL) is a language used to describe object interfaces (attributes and methods) independently of the object oriented languages used for their implementation. From a description of an object in (CORBA) IDL, the compiler generate in a given language both the client object skeleton and the server object skeleton. These must be then complemented by the developer with the specific code for their implementation.

The Object Request Broker from a developer stand point is a set of services supporting the software bus providing location transparency.

8.2.3 CORBA benefits

CORBA provides a common framework to develop applications of an object distributed system which operate in heterogeneous environment.

For bibliographical references and web sites consult the appendices.

8.3 Unified Modeling Language

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

8.4 Definitions

- Metamodel : A metamodel is a language for specifying a model, in this case an object model. In other words, it is a model of modeling elements. The purpose of the UML metamodel was to provide a single, common, and definitive statement of the syntax and semantics of the elements of the UML.
- Use Case diagram : A use case diagram shows the relationship among actors and use cases within a system. A use case is a coherent unit of functionality provided by a system or class as manifested by sequences of messages exchanged among the system and one or more outside interactors (called actors) together with actions performed by the system. Use cases are mapping the User Requirements.
- Class diagram : A class diagram is a collection of (static) declarative model elements, such as classes, interfaces, and their relationships, connected as a graph to each other and to their contents.
- Behavior diagrams : Three types of behaviour diagrams exist : activity diagrams, statechart diagrams and interaction diagrams.
- Statechart diagram : A statechart diagram shows the sequences of states that an object or an interaction goes through during its life in response to received stimuli, together with its responses and actions. They are only needed for those classes of objects that present a sufficient dynamic behaviour.
- Activity diagram : An activity model is a variation of a state machine in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations. An activity diagram is a special case of a state diagram in which all (or at least most) of the states are action states and in which all (or at least most) of the transitions are triggered by completion of the actions in the source states.
- Interaction diagrams : A pattern of interaction among objects is shown on an interaction diagram. Interaction diagrams come in two forms based on the same underlying information but each emphasizing a particular aspect of it: sequence diagrams and collaboration diagrams.
- Sequence diagram : A sequence diagram shows an interaction arranged in time sequence. In particular, it shows the objects participating in the interaction by their

"lifelines" and the messages that they exchange arranged in time sequence. Sequence diagrams show the explicit sequence of messages and are better for real-time specifications and for complex scenarios.

- Collaboration diagram : Collaboration diagrams show the relationships among objects and are better for understanding all of the effects on a given object and for procedural design.

- Implementation diagrams : Implementation diagrams show aspects of implementation, including source code structure and run-time implementation structure. They come in two forms : *component diagrams* show the structure of the code itself and *deployment diagrams* show the structure of the run-time system. Component diagrams depict the software architecture and Deployment diagrams depict the hardware architecture.

- Component diagram : A component diagram shows the dependencies among software components, including source code components, binary code components, and executable components.

- Deployment diagram : Deployment diagrams show the configuration of run-time processing elements and the software components, processes, and objects that live on them. Components that do not exist as run-time entities (because they have been compiled away) do not appear on these diagrams ; they should be shown on component diagrams.

8.4.1 UML basics

A modeling language must include:

- Model elements - fundamental modeling concepts and semantics : the semantics are described in English prose.

- Notation - visual rendering of model elements (see below for overview of the different types of diagrams).

- Guidelines - idioms of usage. It consists of well-formedness rules - The rules and constraints on valid models are defined. The rules are expressed English prose and in a precise *Object Constraint Language (OCL)*. OCL is a specification language that uses simple logic for specifying invariant properties of systems comprising sets and relationships between sets.

In terms of the views of a model, the UML defines the following graphical diagrams:
1. use case diagram (visualizing the User Requirements)
2. class diagram (visualizing the Static Model of OO-Analysis)
3. behavior diagrams (visualizing the Dynamic Model of OO-Analysis) :

a. statechart diagram (internal object diagram for classes with very dynamic behaviour only)
b. activity diagram
c. interaction diagrams:
i. sequence diagram
ii. collaboration diagram
4. implementation diagrams (visualizing Object and System Design):

a. component diagram : depicts the software architecture
b. deployment diagram : depicts the hardware architecture

These diagrams provide multiple perspectives of the system under analysis or development. The underlying model integrates these perspectives so that a self-consistent system can be analyzed and built. These diagrams, along with supporting documentation, are the primary artifacts that a modeler sees.

8.4.2 UML benefits

Developing a model for an industrial-strength software system prior to its construction or renovation is as essential as having a blueprint for large building.

Good models are essential for communication among project teams and to assure architectural soundness. We build models of complex systems because we cannot comprehend any such system in its entirety.

As the complexity of systems increase, so does the importance of good modeling techniques. There are many additional factors of a project's success, but having a rigorous modeling language standard is one essential factor.

The Unified Modeling Language (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction.

The developers of the UML had the following objectives in mind during its development:

- Provide sufficient semantics and notation to address a wide variety of contemporary modeling issues in a direct and economical fashion.
- Provide sufficient semantics to address certain expected future modeling issues, specifically related to component technology, distributed computing, frameworks, and executability.
- Provide extensibility mechanisms so individual projects can extend the metamodel for their application at low cost. We don't want users to have to need to adjust the UML metamodel itself.
- Provide extensibility mechanisms so that future modeling approaches could be grown on top of the UML

- Provide sufficient semantics to facilitate model interchange among a variety of kinds of tools.
- Provide sufficient semantics to specify the interface to repositories forthe sharing and storage of model artifacts.

The primary goals in the design of the UML were as follows:

1) Provide users a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2) Provide extensibility and specialization mechanisms to extend the core concepts.
3) Be independent of particular programming languages and development processes.
4) Provide a formal basis for understanding the modeling language.
5) Encourage the growth of the OO tools market.
6) Support higher-level development concepts such as collaborations, frameworks, patterns, and components.
7) Integrate best practices.

8.4.3 UML characteristics

- Controlled iterative development results in shorter development cycles.
- Model-driven development results in increased developer productivity.
- Use-case and business focused development results in improved software quality
- Focus on software architecture and components results in significant software reuse
- Common, standard language – the UML – results in improved team communication
- Reverse-engineering capabilities allow you to integrate with legacy systems
- Component-based development has emerged as the most effective design process thus far.

In the face of increasingly complex systems, visualization and modeling become essential. The UML is a well-defined and widely accepted response to that need. It is the visual modeling language of choice for building object-oriented and component-based systems.

8.5 Object-oriented management

Roles of Personnel in een OO-development environment are substantially different from those in a classical software project.

Domain Experts capture the user and software requirements and acceptance test the final application.

A central role plays the OO-Librarian or Library Manager. He chairs the OO-Configuration Control Board, implements and approves all changes to the overall UML-class diagram of the Project.

Component Providers implement Components (cohesive collections of associated classes) and provide the detailed implementation of these classes. They execute Component Testing and deliver the Components to the Integrator.

Integrators integrate components with each other and with other systems such as a CORBA-server or an ODBMS. These OO-experts design the UML-model of the subsystem and execute the system testing of the subsystem they deliver. They produce the ADD and DDD of the subsystem.

Project Management must take into account these new roles of personnel, manage OO-methodology, and investigate re-use from other/previous projects in order to achieve the full benefits of OO-technology.

The Quality Assurance Manager must be familiar with OO-metrics and apply these to the Project.

8.6 Object-Oriented Programming languages

We intend to give in this section a brief overview of the most common object-oriented languages, their benefits and drawbacks. We also recommend C++ and JAVA as the languages for development of FINDAS-clients.

A suitable OO-language should have the following characteristics :

- possess the basic OO-features
- have a wide industrial user-base now and in the future
- presence of bindings with other technologies (OODBMS, CORBA, etc.)
- availability on the most common professional computer platforms (UNIX, Win NT, Linux)
- capability to serve the needs of the problem domain
- have an extensive set of libraries
- enabling Internet-programming

Additional features which are important are :

- enabling rapid Graphical User Interface development
- enabling easy Internet and Intranet development
- being easy to learn
- being easy to maintain

Only C++ and JAVA fulfill the first set of characteristics. JAVA has as well the additional features.

C++ is superior to JAVA in terms of performance (speed). The performance of JAVA is improving constantly due to improved compilers, JAVA Virtual Machines and faster microprocessors.

Eiffel is a well-structured and designed OO-language which is wide-spread in the financial world in the U.S. It lacks however the broad user-base of JAVA and C++ and the availability of bindings for other items of the FINDAS infrastructure, such as the ODBMS.

Smalltalk is a very well designed language which is however too academic.

Objective C represents a too limited user community.

JAVA and C++ are the only languages for which bindings exist both with the chosen Database Vendor and with CORBA.

C++ was chosen as a language for prototype development. Developing User Interfaces in C++ is however not easy. and time-consuming. Therefore we recommend to use JAVA for GUI-development (as frontend) and C++ (as backend) for a prototype FINDAS-client.

The definition of the language of choice for the Operational FINDAS-system is outside the scope of this document. It will be determined later.

8.7 Object-oriented databases

Object-oriented databases are a relatively recent phenomenon in the world of databases. They promise to handle better complex data structures and at higher performance than their predecessors, the relational databases. The reason is that they don't possess a so-called impedance mismatch (relational versus OO) of software paradigm which relational databases do have.

They have both object-oriented features inheritance, polymorphism, etc.) and general capabilities of database management systems (rollback, backup and recovery, concurrency control, etc.). They have emerged only recently thus the technology is not yet completely mature. A lot of good utilities and features however does already exist and progress is fast.

The Object Management Group (OMG) sets all the standards for ODBMSs. The emerging standard for storing objects is ODMG. ODMG adds database functionality to object programming languages.

The standards includes the definition of an Object Model, the Object Query Language (OQL) for querying an ODBMS, which is very similar in syntax to the Structured Query language (SQL) for Relational Databases. The standard also defines C++ and JAVA-bindings (APIs) for communicating with the database.

ODMG defines as well an Object Definition Language (ODL). ODL is a database schema definition language extension of the OMG Interface Definition Language (IDL). ODL creates a layer of abstraction such that an ODL-generated schema is independent of both the programming language and the particular ODMG-compliant ODBMS.

Appendix A
Technical terms

Active Objects
Objects which have their own thread of control.

Actor
Something external that interacts with the system.

Agent

Class
an implementation that can be instantiated to create multiple *objects* with the same behavior. An *object* is an *instance* of a class.

Class Diagram
Shows a static structure with classes and their definition and relationships.

Daemon
a process which is always ready to receive requests for services.

Dynamic Type
the dynamic type of an entity at some instant of execution is the type of the object to which it is attached. See also *Static Type*.

Embedding
Wrapping with an appropriate object oriented shell of a non object item.

Encapsulation
the internal representation of a class is encapsulated, i.e. not accessible from outside the class.

Entity
synonym for *Object*.

Implementation
(of class) a definition that provides the information needed to create an object, allowing the object to participate in providing an appropriate set of services.

Implementation Diagram
Shows aspects of implementation, including source code structure and run-time implementation structure.

Inheritance
construction of new definition (class) by incremental modification of other definitions.

Instance
See *Class*.

Interface
a description of a set of possible uses of an object.

Locking

Message
See *Request.*

Method
Code that may be executed to perform a requested service.

Object
a combination of a state and a set of methods that explicitly embodies an abstraction characterized by the behavior of relevant requests. A basic characteristic of an object is its distinct identity which is immutable and persists as long as the object exists.

Passive Objects

Persistent Object
an object which life span is longer than the process that creates it. A persistent object exists until it is explicitly deleted.

Polymorphism
the ability to have more than one dynamic types at run time.

Proxy logging

Request
An event consisting of an operation and zero or more actual parameters.

Service
A computation that may be performed in response to a request.

Static Type
the type with which an entity is declared.

Subclass
a class derived by inheritance from one or more *Superclasses.* A subclass is often used to specialize a *Superclass.* For example TRIANGLE, CIRCLE and SQUARE can be subclasses of SHAPE.

Superclass
the class or classes from which one inherits. Often a Superclass is a generalization of the *Subclasses* which derive from it.

Trigger

Type

classifies objects according to a common interface while classes classify objects according to a common implementation.

Use Case

A coherent unit of functionality provided by a system or class as manifested by sequences of messages exchanged among the system and one or more actors together with actions performed by the system.

Use Case Diagram

Shows the relationship among actors and use cases within a system.

View

Shows different aspects of the system being modelled. A view is an abstraction containing a number of diagrams.

Appendix B
Bibliography

Instant CORBA by *Orfali, Harkey, Edwards* published by *Wiley Computer Publishing*

Client/server programming with Java and CORBA by *Orfali, Harkey* published by *Wilcy Computer Publishing*

Inside CORBA by *Mowbray, Ruh* published by *Addison-Wesley*

Objects on the web by *Ben-Natan* published by *McGraw-Hill*

Object Oriented Software Construction by *Meyer* published byPrentice Hall

UML Toolkit by *Eriksson, Penker* published by *Wiley Computer Publishing*

UML in a nutshell by *Sinan Si Alhir* published by *O'Reilly*

C++ Object databases programming with the ODMG standard by *D. Jordan* published by *Addison-Wesley*

Appendix C
Web sites

- The Object Management Group (OMG) site is at

  `http://www.omg.org`

- A site with many links on object topics is

  `http://www.csioo.com/cetusfr/software.html`

- IONA site is at

  `http://www.iona.com/`

- O_2 information can be found at

  `http://www.ardentsoftware.com/`

- The Rational home page is at

  `http://www.rational.com/index.html`

- The free CORBA page

  `http://adams.patriot.net/~tvalesky/freecorba.html`