# APM

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE • CB3 0RD UNITED KINGDOM
+44 1223 515010 • Fax: +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

**ANSA Phase III**

# Design and Implementation of Quartz

**Zhixue Wu and Toby Speight**

**Abstract**

Quartz, an ObjectLab project, is about service distribution through electronic mail and the World Wide Web (WWW). The aim of the project is to make CORBA objects widely and easily accessible from desktops and consumers, making them simple to use and removing the need to learn a particular interface for each.

This document describes the project principles, design decisions and the implementation.It also presents a demonstration example to show the feasibility and benefits of the project.

APM.1765.01 **Approved** 22nd August 1996
External Paper

**Distribution:**
**Supersedes**:
**Superseded by**:

# Design and Implementation of Quartz


# Design and Implementation of Quartz

Zhixue Wu and Toby Speight

APM.1765.01

22nd August 1996

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by APM Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

APM Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

| | |
|---|---|
| TELEPHONE UK | (01223) 515010 |
| INTERNATIONAL | +44 1223 515010 |
| FAX | +44 1223 359779 |
| E-MAIL | apm@ansa.co.uk |

# Contents

# 1 Introduction

This paper describes Quartz — the results of an APM Object Lab project. Quartz is a system for publishing and accessing CORBA applications via the World Wide Web (WWW), including a simple workflow tool for linking application steps to sequences of Web forms, and a pilot “Electronic Leave Request” application.

## 1.1 Motivation

Since enterprises rely more and more on computer-provided information, so they regard their IT applications as services provided to users, rather than as applications installed and run by those users. As a world-wide electronic marketplace develops, these application service users are located far away from the central site and are equipped with a variety of different types of machine. Installing applications and data is a costly process, especially if they are not to be used frequently or if complete consistency has to be achieved across all sites. Often external users, such as customers of the company or corporate partners, are involved and there is an increasing need for this temporary use of an application or for a punctual connection to a service.

The real requirement is to send the user both the information and its associated applications software but there are still difficulties for local staff in installation, using it and management. Therefore, a simple-to-understand method is required for sending software and information to a user wrapped together as a service with an easy understand, familiar interface.

The Quartz system solves this problem, making application services widely and easily accessible from desktops and consumers, making them simple to use and removing the need to learn a particular interface for each.

## 1.2 The approach

Tools such as electronic mail and the World Wide Web (WWW) are familiar to almost everyone. Their simple and user-friendly interface makes users (even non-technical ones) able to access, provide, and exchange information easily and widely. However, at present they are mostly used for delivering pre-existing HTML or text documents.

The aim of Quartz is to show that these tools can also be used to make application services widely and easily accessible from various platforms. The core idea is to wrap application services with a Web interface, thus removing the need for users to learn a particular interface for each application service, and use email and Web infrastructure for delivering information and services, thus saving time and costs of installation and connection.

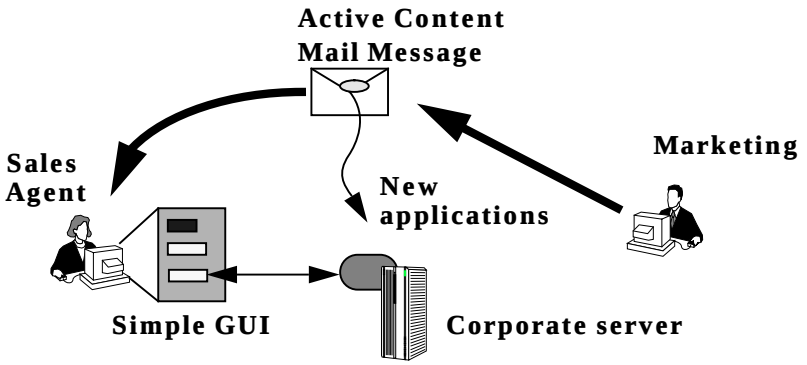
The strategy for achieving this objective is to enable seamless interoperability between WWW and distributed object systems based on the OMG’s Common

Object Request Broker Architecture (CORBA) [OMG 93]. This will make CORBA-based systems accessible to Web browsers. Since CORBA is the leading industry standard for system integration, it provides a pathway for linking the Web to a wide range of electronic information and business services.

## 1.3 Scenario

Figure 1.1 illustrates an idealised scenario for a Quartz system. A marketing agent sends a sales agent an active content mail message that includes an object pointer to a new application service at the corporate server. After receiving the message, the sales agent accesses the new application service directly from the message via a Web browser.

**Figure 1.1: A scenario**




## 1.4 Benefits

By making use of Email and WWW as a simple and easily understood method to provide application services available to users, Quartz provides the following advantages:

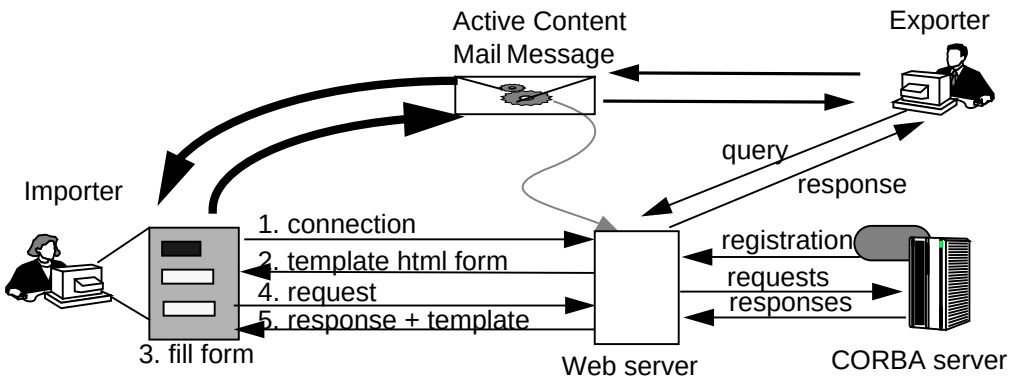
- Accessible from different sites and different platforms
- Easy connection and installation
- Simple but powerful interface
- Easy access to related information
- Rapid application development
- Re-usability and extensibility

# 2 The Configuration

## 2.1 The execution pattern

Quartz can be used for various kind of applications. However, a novel one is to use it as a method of sending software and information to a user wrapped together as a service. A typical use procedure for such kind of application is shown in Figure 2.1.

**Figure 2.1: The execution pattern**




After the service provider publishes a service by packaging it into a Web server, it is available for use. If an exporter, say a marketing agent, requests an importer, say a sales agent, to do some work that requires a new application software, he first obtains the object pointer to the application service by making a query to the Web server. Then he sends an active content mail message that describes the work need to be done, and more importantly includes the pointer to the application service.

After receiving the message, the importer first makes connection to the server by using the object pointer passed from the message. The server will return a client interface for the service in a HTML form with instructions. The importer then can request the server to provide a service by filling and sending back the form. The server, after processing the request, will send a response to the importer. It may also be in HTML format. If more information is needed from the importer to complete the service, a further template HTML form may be passed to the importer together with the response. This procedure will repeat until the work is done.

Finally, the importer notifies the exporter that the work has been done by sending a reply message including the required information.

In this way, the importer, from the exporter's message, not only knows the work needs to be done but also gets the service to do the work. There is no need for her to find or install the software. There is also no need for her to learn a new interface for using the service since the service can be used through the familiar Web interface.

## 2.2 The key issue

Conventional Web servers are document-centric —they are designed to respond to requests for static documents from fixed locations. Therefore, to enable users to invoke CORBA services directly from HTML forms via WWW, extensions to Web servers must be made so as to enable WWW to provide support for connecting live, or interactive services. A key issue, therefore, is to provide a seamless interoperability between CORBA and WWW.

## 2.3 The options

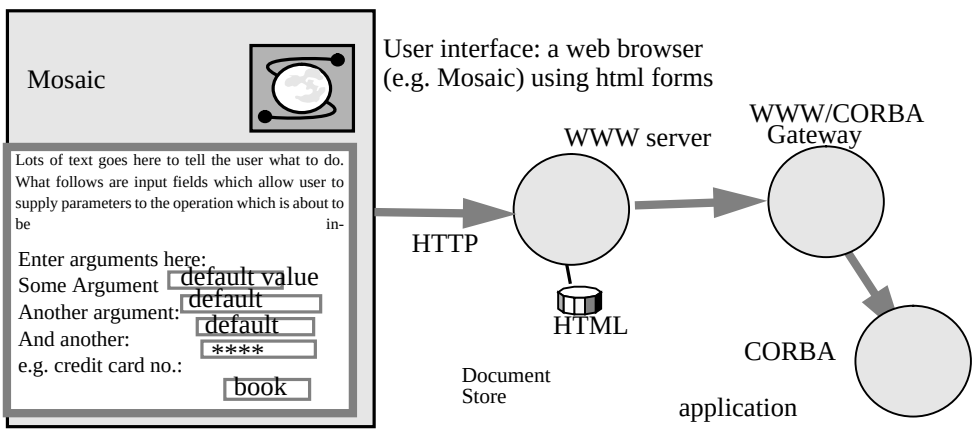
### 2.3.1 Common gateway interface

Common Gateway Interface (CGI) [McCOOL] is a standard for external programs, such as gateways, to interface with HTTP servers. This allows WWW to add third party sources, such as CORBA services, to the system. CGI defines the format of the data stream between the WWW server and gateway, and also the environment variable to the gateway. Bespoke clients for the third party information services are implemented by HTML forms.

Most present Web servers use some form of the CGI mechanism to produce and present documents on the fly in order to respond to requests from the client which contain configuration information.

The advantage of CGI is its simplicity, since its design preserves the stateless nature of Web server transactions by starting a new process for each invocation and passing the Web server environment to that process along with the parameters of the request invocation.

**Figure 2.2: Using CGI to provide interoperability**



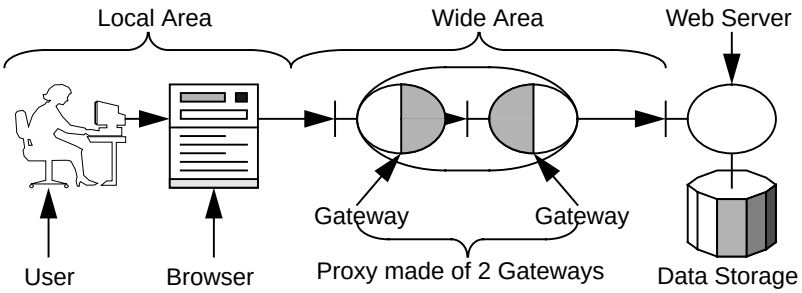

The principal disadvantage of using the existing CGI mechanism for server-side processing is its effect on server performance can be detrimental, due to the high overhead incurred by the need for creating a new process and allocating its resources before the client's request can begin processing. Since it is probably still true that the majority of Web servers are in the business of delivering pre-existing HTML or text documents, this inefficiency contributes little to performance constraints. However, as more and more Web servers begin to deliver response documents that are created, processed, and wrapped on-the-fly, the existing CGI mechanisms will no longer suffice, because they will be the cause of server performance constraints.

Another major disadvantage of CGI starting a process to service each request is that the maintains of state across requests is complicated, normally requiring the use of persistent storage with the attendant extra overhead. Commercial operations are particularly prone to encountering this disadvantage, and the problem is made even worse when the maintenance of state is required by security reasons.

### 2.3.2 ANSAweb

The ANSA team at APM has developed a technology for integrating CORBA and the WWW called ANSAweb.The essence of ANSAweb is the idea of joining the CORBA and WWW worlds by developing protocol gateways between them. The approach taken was to consider HTTP (the WWW Hypertext Transfer Protocol) as the protocol for talking to WWW servers and browsers, but use IIOP (the CORBA Internet Inter-Orb Protocol) as the main transport and invocation protocol wherever possible. The way this is achieved is by exploiting the existing WWW proxy mechanism, as shown in Figure 2.3.

**Figure 2.3: ANSAweb approach**




The greatest advantage of this approach is that it does not require any change to existing WWW servers or browsers. In fact, it is rather like using object wrapping to treat the WWW as a legacy system from the point of view of CORBA.

Since the possibility exists that a service may not be accessible using IIOP, the ANSAweb client-side gateway uses the trading mechanism in the form of a Locator to find an IIOP path to a given server or URL (Uniform Resource Locator). When there is no IIOP service available for a given URL, the system can fall back to HTTP.

The gateway mechanism employed in ANSAweb also fits well with the organizational security model of having a trusted Intranet connected to the untrusted Internet via a gateway or firewall.

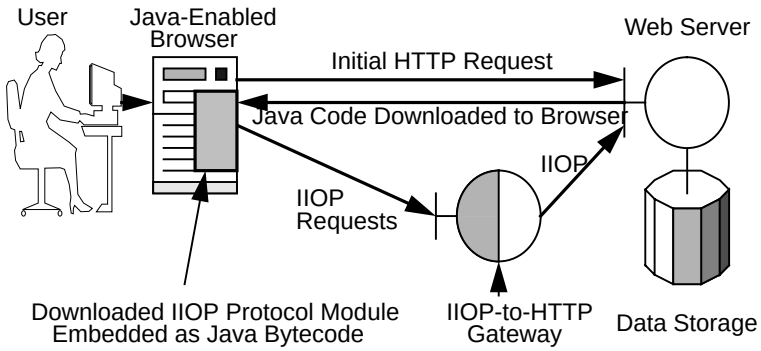
### 2.3.3 Java-based approach

Java was originally designed to facilitate the development of impended system software but has been adapted as a language for WWW programming because of its ability to simplify the development of flexible, portable applications with high-level graphical user interfaces. While offering tremendous flexibility for application development, Java does not, by itself, support a client/server paradigm. In order to support a richer range and scale of application, therefore, a combination of Java with a standardized approach and framework

for application interworking across diverse, heterogeneous systems is required. CORBA provides the crucial missing link between the Java application running on a consumer device and the required service.

The scenario of Figure 2.4 shows a service provider offering a CORBA based service developed using their preferred technology. They then build a custom user interface for the service as a Java applet. The applet makes use of the Jade IIOP module to talk to the CORBA service. The user interface is then made available on the WWW, along with the advertisement for the service. When a consumer accesses the service for the first time, their Java enabled browser automatically downloads and auto-installs the user interface applet. From then on, the consumer accesses the service through its own custom interface.

**Figure 2.4: The Java approach**




#### 2.3.3.1 *Jade: Java applets over an IIOP module*

The Jade project at APM is about service access and provision on the internet. The essential idea behind Jade is to use the new generation of Java enabled Web servers and browsers as base platforms for the insertion of an IIOP package as a Java module.

#### 2.3.3.2 *Java RMI*

Java Remote Method Invocation (RMI) is a Java-specific means of distributing Java objects on a network. It is not a CORBA-compliant solution; rather, it is optimised for the Java language, and interfaces must always be defined in Java (not in a language-independent fashion such as CORBA IDL).

#### 2.3.3.3 *OrbixWeb*

Iona's OrbixWeb allows CORBA-compliant Orbix client functionality to be downloaded as a Java library to the client target. This enables Java applets on the target host to interoperate with any CORBA 2.0 compliant service.

#### 2.3.3.4 *BlackWidow*

The BlackWidow Java Object Request Broker developed by PostMondern Computing is a complete CORBA 2.0 compliant implementation written in Java, providing full client-side and server-side functionality. Since the ORB runtime is written completely in Java, Applets using BlackWidow can be downloaded into a Java enable web browser and can communicate with any other CORBA 2.0 compliant object, running anywhere on the Internet, or within an Intranet.

## 2.4 The choice

Quartz takes the Java-based approach to provide interoperability between the Web and CORBA. The advantage of the Java-based approach is that it allows dynamic bootstrapping of CORBA connectivity. Browsers download the Java IIOP module from a Web server using HTTP.

The advantages of using Java are its platform independence, avoidance of intellectual property problems (since source is not shipped), built in low level security, automatic upgrading and runtime configuration of clients, and a growing third party development base.

The CORBA Java integration is of significant strategic importance in the development of Internet business as it combines the flexibility and ease of use of the Java environment with a standardized approach and framework for global object interaction.

The Java RMI could be used, but this would constrain server systems to be written in Java, or at least provide a Java gateway.

The usual security policies of current Java browsers prevent them from accessing hosts other than their source, so it is necessary to provide the servers on the same host as the Web server, or provide a proxy at that point. The use of a proxy also makes it easier to develop systems which may have to cross administrative boundaries protected by firewalls.

# 3 Workflow on the Web

To demonstrate the feasibility, advantages and benefits of the idea, Quartz has developed an application example, namely a workflow management system. This is described in this section.

## 3.1 Workflow applications

A workflow consists of a collection of activities which support a specific business process. Classical examples range from claim processing in an insurance company to production scheduling in a manufacturing company to leave request management within a company.

A typical character of workflow applications is that information flow across multiple desktops which may be in various platforms and in a wide range of places. Therefore, it requires platform independence and easy connection.

Another character of workflow application is that usually a third party would be involved temporally in a process. For example, when an insurance company is dealing with a claim, it is likely to need to cooperate with another insurance company. The third party may require temporary access to the system. This requires punctual connection and easy to understand interface.

Workflow applications usually need to access other information systems to get required data to make decisions. For example, to grant a leave request, a manager need to check the company's diary to see whether there is a collision between the requested time and some important company activities.

Quartz provides exactly the technology to meet these requirements. It can make an application service available easily and widely for any platform. It provides an easy-to-understand and familiar interface to an application service and can easily interface to existing information systems. More importantly, by using ANSAweb or Jade technology, it also enables interactive communication between Web browsers and Web servers, the main disadvantage of the conventional Web technology.

## 3.2 The architecture

The workflow management system architecture (Figure 3.1) is based on active rules. Rules are generated from workflow specifications providing the operational semantics of workflow enactment and giving a concrete option for implementation.

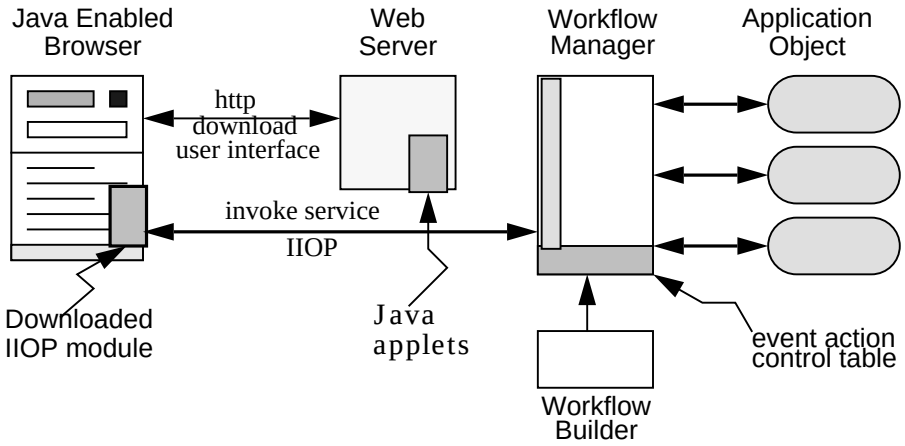
### 3.2.1 The workflow manager

The workflow manager is the management and administrative component of the workflow management system. It actively intervenes to keep every process on track. It monitors and controls the process of a workflow according to the

specified active rules. When an event happens, it will command certain application objects to take appropriate actions.

Workflow manager component is independent from particular workflow applications. Thus, it is desirable to implement it in an application-independent form so that it can be reused in different workflow applications, or even shared by concurrently running applications. Of course, the active rules are application specific and are different for different workflow applications.

**Figure 3.1: The architecture of a workflow management system on the Web**




### 3.2.2 The workflow builder

The workflow builder is a mechanism for workflow developers to describe the active roles and translate the active roles from a user format to a internal format which is understandable to the workflow manager.

The purpose of the workflow builder is to provide an easy way for workflow application developers to describe the active rules according to their application semantics. Generally, a process of a workflow can be specified in a finite state machine by describing what actions should be taken when an event happens in a particular state and what state the workflow should be transferred to.

### 3.2.3 Application objects

Application objects implement actions for a workflow application. They are application specific, i.e. implemented for a particular workflow application. Application objects are registered to the workflow manager through active roles. They are generally passive in the sense that they only take actions according to the command from the workflow manager.

### 3.2.4 User interface

A user starts a workflow by specifying their request and providing required data through a Web-based user interface to a workflow management system. The user interface also provides methods to query the state of a workflow or the system.

The function of the user interface is to collect request and data from a user, and pass them to the workflow manager by using the API of the workflow

manager. It hides the API from users and provides them with an easy-to-use interface.

Like application objects, a user interface is also application specific. The data that should be requested from a user is dependent on the requirements of the application objects.

Since Quartz takes the Java-based approach for providing interoperability between Web and CORBA, a user interface for a workflow application is implemented as Java applets using Java's API as well as the API of the workflow manager.

The applets are then embedded into HTML forms which will describe the functionality of the applets. They can be downloaded by interested users to their Java enabled Web browser and access the workflow application service through them.

### 3.2.5 The Web server

The web server in the architecture is twofold. First, it is used for distribute a service. Java applets for both the IIOP engine and the workflow client interface are stored with a Web server and published in the Web so that users can download the applets to use the service provided by the workflow management system.

Secondly, it can also be used directly in a workflow application, for example, to provide on-line help, or to provide means for users to check related information during the process for making a request and inputting data.

### 3.2.6 The Web browser

A Java enabled Web browser is used as for the user interface. After fetching the Java applets for a workflow application, users can use the service through a Web browser. When the workflow application service is accessed at the first time, the IIOP engine applets developed by the Jade project will be automatically downloaded to the user site and thus can be used for connecting the Web browser to the workflow system.

## 3.3 The Implementation

### 3.3.1 The workflow builder

A GUI is an ideal tool for the administrator to specify the active rules for a state machine. However, we only provide a small script language for specifying active rules. A GUI could be added to generate this language, and the language could also be extended with a small amount of work. A more advanced workflow builder could also generate client stubs (as either HTML forms or Java classes); this gives similar consistency benefits as the use of IDL to generate CORBA client stubs and server skeletons.

The active roles define all events that allow to happen in a workflow application, and what actions should be taken when an event happens. For an event, it may be allowable only in certain states. Therefore, to describe an event, one need also specify in which state it is allowable and which state the workflow should transfer to after the event.

Figure 3.2 shows a specification for an event called **employee_sub.** The event has 3 **in** arguments. It is allowable when a workflow is at state 1. The event will transfer the workflow state to 2 afterwards.

When an **employee_sub** event happens, two actions will be taken: a **newForm** action and a **fillForm** action. To describe an action, one need to specify the object which the action will be applied to by giving its name, type, and host machine.

**Figure 3.2: A specification of an event**

```
event = employee_sub;
in_args_no = 3;
inout_args_no = 0;
out_args_no = 0;
applied_state = 1;
next_state = 2;

action = newForm((0,0));
objectName = ApplicationForm;
objectType = FormBase;
hostName = stardust;

action = fillForm((0,0), (0,3));
objectName = ApplicationForm;
objectType = FormBase;
hostName = stardust;
```

### 3.3.2 The workflow manager

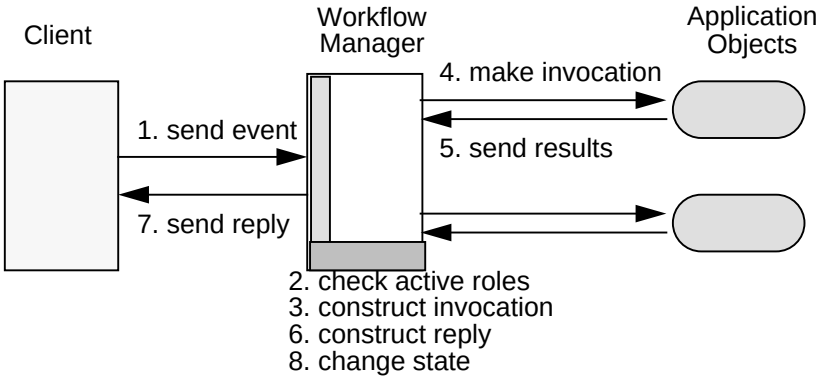
The key issue for implementing the workflow manager is how to make it application-independent so as to make it re-usable for various workflow applications. Quartz resolves the issue by making use of the Dynamic Invocation Interface (DII) of a CORBA system.

The workflow manager provides two operations for its clients. An operation for starting a workflow: **start_workflow**, and an operation for sending an event: **send_envent**. When starting a workflow, a client needs to give a name to the workflow. The name is used to identify a workflow from others and is always required for annocing an event.

When announcing an event, a client needs to specify, besides the workflow name, the name of the event, and the values and modes of their arguments.

The procedure of processing an event is shown in Figure 3.3. When receiving an event, the workflow manager will first match the event at the active roles table. If the event is defined and it is applicable at the current state of the named workflow, the workflow manager will use the data provided in the event to construct appropriate operation invocations according to the active role. Then the workflow manager will make each constructed invocation to appropriate objects. When the results from all the invocations come, the workflow manager constructs a reply to the event based on the results and sends the reply to the client. Finally, the workflow manager changes the state of the workflow to their next state.

**Figure 3.3: The procedure of processing an event**




The workflow manager is implemented in C++ on Oribix 2.0, a CORBA 2.0 implementation from Iona.

# 4 The Electronic Leave Request System

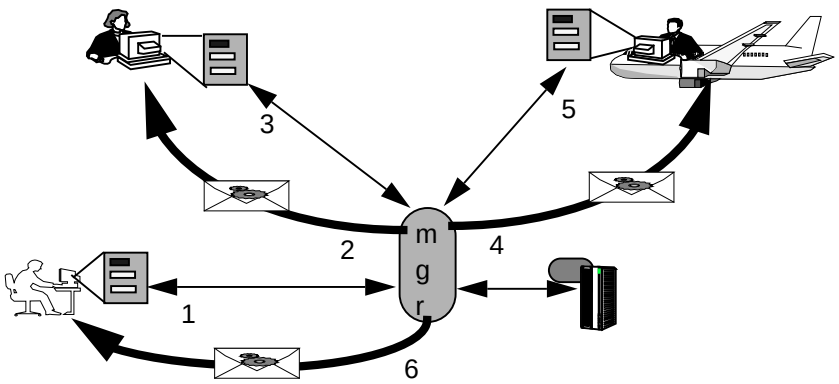
To show the feasibility and the benefits of Quartz, an application has been developed, namely an electronic leave request system. This is described in this section.

## 4.1 A scenario

In any organisation, there is a system for staff to book their leave or holiday time. Usually it is manually implemented, and some paper work is involved. The forms and documents are manually routed from one person to another. To track the status of an application, one has to go around and ask where it is. By providing an electronic leave request system, this procedure is automated, thus can be done quickly and easily.

The following is a typical scenario the electronic leave request system has to support (see Figure 4.1):

**Figure 4.1: A scenario of the leave request system**




1. A member of staff makes a leave request by filling an application form through a Web browser, say on a Unix workstation.
2. After receiving the submission, the workflow manager notifies the secretary about the application by sending an email including the application form.
3. The secretary, perhaps working on a PC, reviews the application form encapsulated in the email, and checks whether the employee has enough available holiday time.
4. Supposing the employee is entitled to the amount applied for, the workflow manager sends the manager an email about the application.
5. The manager (who may be away for a conference, and using a laptop machine) reviews the application form via a Web browser. He will decide whether to accept the request, checking whether there is any time

collision with important company activities by reviewing the company diary through a Web Browser.

6. Finally, the workflow manager sends an email to the staff to notify the result.

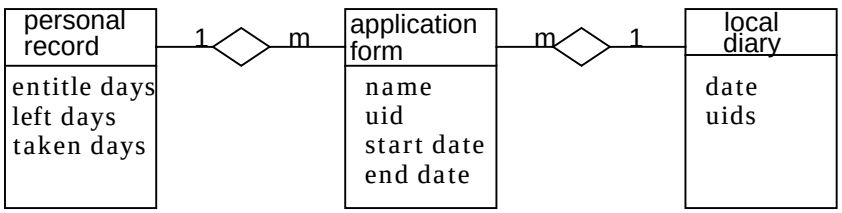
## 4.2 Application Objects

### 4.2.1 Entities and their relationship

A leave request system not only deals with application forms from staff, but also needs to cope with other related information, such as personal holiday record document, that records the days an employee is entitled to, the days taken and the days left, and a local diary, that shows who will be away in which days. To reduce the typing work and maintain consistency between them, it is desirable for the leave request system to update related documents automatically when a leave request is granted.

Therefore, there are three entities involved in a leave request system: an application form file, a personal holiday record file, and a local diary file. Their relationship is shown in Figure 4.2.

**Figure 4.2: The entity relationship**




Whenever an application is accepted, the workflow manager will automatically update the personal holiday record file of the applicant, and also the local diary file. Thus, they will always be consistent with each other.

### 4.2.2 The application form object

The IDL definition for the application form is shown in Figure 4.3.

**Figure 4.3: IDL for the application form object**

```
// ApplicationForm.idl
#include “Form.idl”

interface ApplicationForm {
// IDL operations
void view(out Form applForm);
void fill(in Form applForm);
void update(in Form applForm);
};

interface FormBase {
ApplicationForm newForm(in string name);
void deleteForm(in string name);
void viewForm(in string name, out Form applForm);
void fillForm(in string name, in Form applForm);
void updateForm(in string name, in Form applForm);
};
```

The ApplicationForm object has three operations that can be used to access a form: view, fill and update. An ApplicationForm object is managed in a database called FormBase. The FormBase object can produce a new form, delete an existing form, and view, fill or update a particular form in the database.

### 4.2.3 The personal holiday record object

The IDL definition for the personal holiday record is shown in Figure 4.4:

**Figure 4.4: IDL for the personal holiday record object**

```
//PersonalRecord.idl

interface PersonalRecord {
// IDL operations
void view(out Pform pForm);
 void fill(in Pform pForm);
 void update(in Pform pForm);
 void updateDays(in Form aForm);
};

interface RecordBase {
 PersonalRecord newRecord(in string name);
 void deleteRecord(in string name);
 void viewRecord(in string name, out Pform pForm);
 void fillRecord(in string name, in Pform pForm);
 void updateRecord(in string name, in Pform pForm);
 void updateDays(in string name, in Form aForm);
};
```

Like an application form object, an personal holiday record object is managed in a database called RecordBase. The RecordBase object can create a new personal holiday record object, and delete an existing record. It can also fill, update and view a particular record in the database.

### 4.2.4 The email delivery object

Email system provides an excellent and almost instantaneous means of routing information. The function of the email delivery object in the leave request system is to inform the manager, or the secretary that an application form needs to be processed, as well as to tell them the location of the application form and the software to process the application form. Email is also used to inform an applicant about the result of the application.

The IDL for the email delivery object is shown in Figure 4.5.

**Figure 4.5: IDL for the email delivery object**

```
//email.idl

interface Email {
//IDL operations
void toSecretary(in string wkflw, in string url, in string addr);
void toManager(in string wkflw, in string url);
void toEmployee(in string wkflw, in string url);
};
```

## 4.3 Data persistence

An important requirement from practical application, such as the leave request system, is persistent objects -- long lived objects stored on disk in the file system or a database. For example, an application form object needs to be persistent so that it can be reviewed by the secretary and manager at some later stage after a staff creates it.

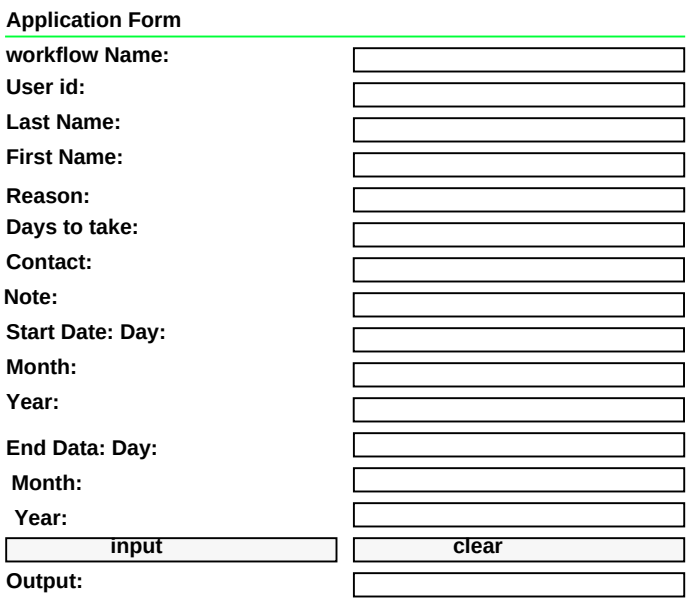
Orbix does not by itself provide persistent objects, but provides users a Loader mechanism which can be used for enabling Orbix objects to be stored and then loaded when they are next required. When an operation arrives at a process, Orbix searches for the target object in the process's object table. By default, if the object is not found, Orbix returns an exception to the caller. However, if one or more loader objects are installed in the process, these will be informed about the object fault and provided with an opportunity to load the target object and resume the invocation transparently to the caller.

Quartz support persistent objects by providing its own loader which stores an objects to (when a process terminates), and reads an object from, (when a object fault occurs), normal files. Each object is stored in its own file and uses its marker as the file name.

## 4.4 The client interface

The user interface to the electronic leave request system is HTML forms with embedded Java applets. The functionality of an applet is described in the HTML form. Users choose the appropriate applet to download according to these descriptions. A Java applet itself is also an GUI with instructions so that there is no need for users to learn a particular interface to use the system. Figure 4.6 shows the appearance of an application form for a user to book a holiday.

**Figure 4.6: A typical user interface**



# 5 Delivables

The Quartz project produced the following results

- A working demonstration example to show the feasibility, advantages and benefits of its technology, namely the electronic leave request system. There are two versions of the demonstration. One takes the Java-based approach; the other takes the CGI-gateway approach.
- A workflow manager object that can be reused in similar applications. The workflow manger object is carefully designed and implemented so that it can be reused without making any change to it. Since it is the key component for workflow applications, its reusability will provide great benefits to such kind of applications.
- A simple workflow builder that can be used to specify active roles for workflow applications.
- A project report (i.e. this paper) that describes the project principles and design decisions, as well as the key architectural features of the demonstration example.

# 6 Summary

The Quartz system provides a simple-to-understand method to make CORBA services available easily and widely by using familiar tools such as electronic mail and WWW, and advanced object technologies such as Java and CORBA. The feasibility, advantages and benefits of Quartz system has been shown in a demonstration application example, namely an electronic leave request system. Although the demonstration is a workflow application, its principle and technology can be applied to a wide range of applications.

Because the main purpose of the project is simply to prove the idea and to demonstrate its advantages, and because the project was done to a time limit, there are some properties that are important for practical applications, but have not be provided in the demonstration applications. Support for concurrency control and transaction mechanisms is necessary for any system that is shared by multiple users. Another important issue for the leave request application is authorisation and access control. For example, a member of staff perhaps should not allowed to access the information of other employees. Although Quartz does not provide these properties, its approach does not cause any difficulty in their support.

# References


[IONA 96]

"*OrbixWeb White Paper*", IONA Technologies Ltd., February 1996, <**URL:http://www.iona.com/Orbix/Java/**>

[JAVA]

JavaSoft team, "*Java documentation*", Sun Microsystems, Inc., <**URL:http://www.javasoft.com/doc/**>

[McCOOL]

Rob McCool, "*The Common Gateway Interface*", <**URL:http://hoohoo.ncsa.uiuc.edu/cgi/overview.html**>

[McCLENAGHAN 96]

Ashley McClenaghan & Youcef Laribi, "*Jade Project Overview*", <**URL:http://iiop.ansa.co.uk:8080/~jade/Documentation/ProjOver/**>

[OMG 93]

"*The Common Object Request Broker: Architecture and Specification*", Revision 1.2, Draft, December 1993, OMG Document Number 93-12-43. <**URL:ftp://ftp.omg.org/pub/docs/93-12-14.ps.z**>.

[POMOCO]

Visigenic, *VisiBroker for Java* (formerly PostModern Computing's *Black Widow*), <**URL:http://www.visigenic.com/BW/bwhome.html**>

[REES 95]

Rees et al, "*A Web of Distributed Objects*", WWW Journal 1(1):75-87, 1995. <**URL:http://www.ansa.co.uk/ANSA/ISF/wdistobj/Overview.html**>

[WfMC]

Workflow Management Coalition <**URL:http://www.aiai.ed.ac.uk/WfMC/**>