The Comandos Distributed Application Platform

Vinny Cahill, Roland Balter, David Harper, Neville Harris,
Xavier Rousset de Pina and Pedro Sousa

Vinny Cahill and Neville Harris
Department of Computer Science,
Trinity College Dublin,
Ireland
Phone: +353 1 7021795
Fax: +353 1 6772204
Email: Vinny.Cahill@dsg.cs.tcd.ie

Roland Balter and Xavier Rousset de Pina
Unite mixte BULL-IMAG,
Gieres,
France
Phone : +33 76 54 49 12
Fax : +33 76 54 76 15
Email: Roland.Balter@imag.fr

David Harper
The Robert Gordon University,
St. Andrew Street, Aberdeen AB1 1HG
United Kingdom
Phone: +44 224 262701
Fax: +44 224 262727
Email: David.Harper@robert-gordon.ac.uk

Pedro Sousa
INESC,
R. Alves Redol 9, 1000 Lisboa,
Portugal
Phone : +351 1 3100287
Fax : +351 1 525843
Email: pms@inesc.inesc.pt

Abstract

This paper presents an overview of the Comandos distributed application platform. It begins by presenting the scope and objectives of the platform before introducing its main concepts, design choices and overall architecture. An overview of the various prototype implementations of the platform undertaken within the Comandos project is also presented. In addition, the paper provides an introduction to the other papers on Comandos in this issue and sets the context for the research reported therein.

Published in *The Computer Journal*, vol 37, no 6, August 1994. Also technical report TCD-CS-94-40, Dept. of Computer Science, Trinity College Dublin.

The development and integration of application software is currently a labour and cost-intensive proposition, particularly for cooperative applications in which large volumes of distributed structured data are shared by cooperative users. Methodologies and tools are needed to master the complexity inherent in the use of heterogeneous distributed environments and in new application requirements.

The overall objective of the Comandos project[1] [Cahill *et al* 1993] which ran from 1986 to 1993 was to specify and construct an integrated platform for programming and managing multi-vendor distributed systems. The platform was to be targeted at application programmers and system administrators with the aim of reducing the overall cost of the development, maintenance, and evolution of large distributed applications, as well as the reuse of old-style (UNIX[2]) applications. The platform was expected to be a basis for integrated information systems in application domains such as office and business systems, computer aided design and software engineering.

Comandos was to support the development of integrated (also described as tightly-coupled) distributed applications within a *cell* which constitutes the basic organisational and administrative component within an enterprise computing system. A cell is typically composed of a set of cooperating workstations, servers and processor pools connected through a high-speed local area network. The goal of Comandos was to present the distributed system to its users as a coherent entity despite the variety of its components.

This paper gives an overview of the main concepts, design choices and overall architecture of the resulting distributed application platform as well as introducing its various prototype implementations. The paper also serves as an introduction to the other papers on aspects of Comandos to be found in this issue.

[1]This work was partially supported by the Commission of the European Community under Esprit contracts 834 and 2071.

[2]UNIX is a trademark of UNIX Systems Laboratories, Inc.

The paper is structured as follows. Section 1 introduces the Comandos model which describes the functionality provided to application developers and system administrators. Section 2 describes the architecture of the platform and introduces its various implementations. Section 3 introduces the other papers in this issue, while section 4 contains a summary and some conclusions.

1 The Comandos Model

The Comandos project adopted an innovative approach based on the integration of operating system, programming language and database technologies. A unified view of the platform is provided by a model and system architecture based on object-oriented technology coupled with distributed persistent storage in which objects are the units of programming and data modelling, as well as the units of distribution and storage.

This conceptual model of a distributed environment, encompassing both computation and data management, presents the functionality of the Comandos platform to application programmers and system administrators. The model is abstract in the sense that it does not require the use of any particular programming language. Technically it consists of two major components:

- A *computational model* which allows distributed programs to be defined. This model provides the application designer with a multi-node multi-processor virtual machine, in which parallelism is apparent and distribution is hidden.
- Common and extensible *type* and *data models*. The type model captures the type systems of various object-oriented programming languages and provides a basis for supporting cross-language invocation. The data model provides abstractions for modelling collections of objects, relationships between such collections and classification structures.

1.1 Functionality of the Platform

The fundamental goal of Comandos was to make available a platform providing the functionality necessary for the construction and management of sophisticated distributed applications. Thus, the Comandos platform includes infrastructure supporting:

- transparent access to distributed resources and services;
- transparent access to persistent data;
- concurrent distributed computations;
- fault tolerance;
- controlled sharing of resources;
- data management;
- security and protection of data;
- on-line system management, monitoring and control.

In the following sections, the approach taken to supporting each of these features is presented.

1.1.1 Transparency

One of the main requirements on the Comandos platform was transparency, i.e. the provision of a coherent and uniform view of all the resources (including data) and services provided by the distributed system. In Comandos, all resources and services are modelled as objects. Transparency implies that the platform provided to the user hides the distribution of objects and processing, as well as the possible heterogeneity of the underlying hardware. In addition, the platform should provide user mobility, i.e. the ability for a user to access objects independently of being logged on to a specific workstation.

Transparency has many aspects: access transparency, in which both local and remote objects are accessed in the same way; location transparency, in which the location of an object is not apparent from its name; execution transparency, in which the execution site of a program may be easily changed, possibly in a dynamic way; environment transparency, in which the same program has the same effect independently of the site on which it is executed; failure transparency, in which a partial failure may be bypassed, so allowing programs to be fault-tolerant; and finally, performance transparency, in which the costs of remote access are not generally degraded over those of local access.

Comandos supports access, location, execution and environment transparency. Although full failure transparency was not supported, a transaction mechanism (see section 1.1.4) is provided which allows an application to ensure that the data which it is accessing remains consistent even if the execution of the application is interrupted by a failure. Comandos also supports full user mobility.

In some cases, transparency is not desirable, and some applications may elect to be aware of the distribution of the objects which they manipulate. For such cases, the platform primitives which allow the management of object location are made visible to the application programmer.

The central execution mechanism of the Comandos platform is object *invocation*, i.e. the execution of a specified method of an object by a process executing at some node. If the object happens to be present in virtual memory at the node, e.g. as a consequence of a previous invocation, the invocation is performed locally. If the object is not present in local virtual memory, it may be present in virtual memory at some other node, again as a result of a previous invocation. In this case a *remote invocation* would be carried out. Finally, it may happen that no node currently has an image of the object in virtual memory. In that case, the object must be located in secondary storage and would be *mapped* (loaded) into virtual memory at some node where the invocation would then be carried out. The choice of the node is determined by the *execution policy* in force and could depend, for example, on the load on the distributed system.

Note that each object resides entirely on a single node (either mapped in virtual memory or stored in secondary storage). Comandos does not support fragmentation of individual objects. This design choice resulted directly from the granularity of objects expected (usually small) and from the general approach in which large compound structures can be built out of object components. In Comandos, a complex compound object and its various object components may reside on different nodes.

1.1.2 Persistence

In conventional programming languages, the lifetimes of most entities are bounded by the duration of a program run. External files are usually the sole exception, with the consequence that if a particular entity is to survive a program run, it must be inserted into a file (possibly requiring a change in its representation) and explicitly retrieved (and possibly rebuilt) at a later time. The disadvantages of this approach are well known [Atkinson *et al* 1983]. In systems providing persistence, programmers can manipulate persistent entities without explicit I/O.

In Comandos, persistence is defined via the reachability property of objects – that is, every object reachable by recursively enumerating the constituent objects from some specified set of root objects is guaranteed by the platform to persist. Thus, there can be no dangling references to objects.

When not in use, persistent objects are stored on secondary storage. When an attempt to access such an object is made, the object is automatically fetched from storage by the platform and made available in virtual memory at an appropriate node. Hence it is unnecessary for the programmer to write code either to explicitly fetch the object from storage or to convert between the possibly different secondary storage and virtual memory representations of the object since this is handled automatically by the platform. Likewise, when the object is no longer required by any running application, the platform automatically returns the object to secondary storage – there is no need for the programmer to explicitly store the object.

When an object is retrieved from storage other related objects may be retrieved at the same time and in the same I/O operation so that they are available for use should they be required. Such a group of related objects is known as a *cluster*. Such prefetching of objects improves performance by reducing the number of mapping and I/O operations required when related objects are used together.

The paper by Sousa and others in this issue discusses the approach to persistence in the IK implementation of the Comandos platform (see section 2.1.1).

1.1.3 Concurrency

An important design decision was how to relate objects to processes. Two possible solutions were available:

1. Associate processes with objects, i.e. define active objects, in which every object contains a fixed or variable number of processes. This solution was used in previous systems such as Argus [Liskov & Scheifler 1983] and Eden [Almes *et al* 1985].
2. Separate objects from processes, i.e. define passive objects which can be operated on by independently defined processes. This is the solution adopted in Clouds [Dasgupta *et al* 1988], Amoeba [Mullender 1985] and SOS [Shapiro *et al* 1989].

The active object solution is conceptually simpler since a single abstraction encompasses the concepts of resource and process. On the other hand passive objects contain less context information than active objects. Therefore object creation, invocation and migration can be implemented more efficiently than for active objects. Moreover, since the classes of application that Comandos was intended to support typically involve creating many (usually small) objects, and building large compound structures out of object components, the passive object solution was chosen as being the more appropriate.

In Comandos, a *job* represents the processing (possibly in parallel) of objects and consists of one or more sequential threads of control, called *activities*. A job is created by an activity of another job to invoke a specified method on some object. A new job initially consists of a single activity.

The execution of an activity consists of nested invocations of methods on objects. Each invocation may take place on any node in the system. At each invocation the referenced object is located and, if necessary, loaded into virtual memory at some node. A remote invocation takes place if the node selected for execution is different from the node on which the invocation was requested.

An activity may, at any time, create one or more parallel activities within the same job. Therefore the platform provides two levels of concurrency: between activities belonging to the same job (i.e. related computations within the same application); and between activities running in different jobs (i.e. independent applications). Asynchronous invocation can be implemented by creating a new activity (or job) to invoke a designated object.

1.1.4 Atomicity

Atomic transactions [Bernstein *et al* 1987] provide a means of ensuring the consistency of data in the presence of concurrency and partial failure. In particular, transactions classically guarantee certain properties – atomicity, consistency, isolation and durability – for operations carried out within the transaction [Gray & Reuter 92].

In fault tolerant distributed systems it is common sense that the atomicity property of computations should be provided. For some applications consistency of data is also required. However, providing the atomicity property of computations and ensuring the consistency of the affected data are, at least in principle, orthogonal to each other. Consequently, a general backward error recovery mechanism which makes no assumptions about the synchronisation mechanism used should be provided. This enables the use of different and even no synchronisation mechanisms in the future without having to change the underlying recovery mechanism. As a first step in this direction Comandos provides a *generalised transaction mechanism*. This mechanism is based on a recovery layer which is able to manage dependencies between transactions. Thus, in contrast to traditional transactional systems, ensuring consistency of transactions becomes possible without having to enforce the usual restrictive failure isolation property.

The Comandos transaction model is derived from that provided in RelaX [Schumann *et al* 1989]. RelaX provides generalised distributed transactions, extending the classical transaction model by supporting optional use of uncommitted data, extended nesting (allowing the differentiation between recovery and synchronisation levels), and the separation of transaction commitment from completion. A transaction consists of a set of operations on objects which has the properties of atomicity, consistency and durability.

Since transaction mechanisms involve a certain overhead which should not be incurred by all objects, these mechanisms apply only to a subset of objects known as *atomic objects*. An object may be created as an atomic object or a non-atomic object can be promoted to be an atomic object (the reverse not being permitted). Hence the transaction properties are only guaranteed for operations on atomic objects carried out within a transaction. A transaction is contained entirely within a single job.

In addition to accessing atomic objects, transactions may also access other (transactional) resource types such as external (to Comandos) files or database records. This is achieved by structuring the implementation according to the X/Open model [X/Open Company 91]: the Comandos platform acts as a collection of X/Open Application Programs and Resource Managers that interact with the RelaX Transaction Manager [Kröger, Mock *et al* 90] which is responsible for distributed transaction management via an interface which extends the X/Open XA-interface. Pure XA-compliant Resource Managers can still be used.

The paper by Taylor and others in this issue discusses a number of issues related to the design of the transaction support in the Amadeus/RelaX implementation of the platform in detail.

1.1.5 Sharing

There is no explicit communication through message passing between activities. Instead communication and synchronisation between activities (within the same job, or belonging to different jobs) is achieved by invoking on shared objects.

Object sharing may be controlled in a number of ways depending on the consistency requirements for concurrent usage. Atomic objects are guaranteed to remain consistent despite concurrent access and partial failure of the underlying system.

Comandos makes no guarantees about the consistency of objects which are not atomic. Other objects (i.e. their class code) can maintain their own consistency in the presence of concurrent accesses by making use of primitive concurrency control mechanisms – such as semaphores – provided by the platform.

1.1.6 Data Management

Comandos provides services for the management of large collections of objects (including associative access to objects belonging to collections), based on a data model which provides a set of constructs that enables application programmers to model real world entities and the relationships between them with relative ease. The Comandos data model, the Binary Relational Object-Oriented Model (BROOM) [Harper *et al* 1991], provides four kinds of collections which might be considered to be special forms of sets, sequences, bags and binary relations. These collections are provided through a number of generic bulk type constructors.

The inclusion of binary relations as collections is novel and therefore requires some comment. In recent years, the various forms of entity-relationship data models have been popular for data modelling. The basis of this approach is to model the real world in terms of entity sets and relationships between entity sets. Support for the direct representation of relationships is extremely beneficial in data modelling. A deficiency of the object-oriented approach is its inability to do just that. Therefore, the usual notions of the object-oriented data model have been extended to include direct representation of relationships between entity sets: this is achieved by introducing the binary relation as a kind of collection in the Comandos data model.

A further limitation of many existing object-oriented data models is that they allow only one collection of objects of a particular type. Thus, associated with a type there may be a collection which comprises the set of all current instances of that type. In BROOM, the notions of typing and classification have been separated. This permits several collections of objects to be associated with a given type. Collections are related by means of a classification structure. This approach provides a much more flexible modelling capability.

The classification structure is represented by means of structural constraints among collections. For example, an IS-A relationship between two set collections `C` and `D` says that every object that belongs to collection `C` must also belong to `D` and this may be represented by a subset constraint between `C` and `D`. Other forms of structural constraints supported correspond to partitions of collections, the intersection of collections, and cardinality and dependence constraints on relations.

1.1.7 Security

The Comandos platform provides distributed applications with facilities to keep objects secure [Medina & Moreno 1991]. The ultimate goal of security in an object-oriented environment is to enforce the constraint that an object can only ever be manipulated by authorised, authenticated and secure invocations of type-specific operations on that object[3]. The mechanisms provided towards achieving this goal in Comandos are:

- Secure transmission of the arguments and results of each (remote) object invocation, by the use of encryption techniques to prevent playback, copying or modification of messages.
- Authorisation checks on object invocations which are achieved through the use of access control lists associated with each protected object. The access-rights can specify the set of operations that a user (or a group of users) can perform on the object.
- Isolation of objects at run time so as to protect against damage to an object by code of another object executing in the same address space.

In addition an audit service is provided which allows the system administrator to detect intrusion into or misuse of the system [Weiss & Baur 1990].

These facilities are implemented by low-level mechanisms integrated within the platform. High-level management tools for the overall administration of the security policies and procedures (e.g. user and group management, etc.) are also provided.

Note that while no authentication service was provided, it was expected that an authentication service such as Kerberos [Steiner *et al* 1988] could be integrated with the platform.

[3] where direct access to the data of an object, if supported, is considered as a special form of operation invocation.

1.1.8 Management View of the Platform

Most approaches to organisational [Grochla 1982] and systems design [Lockemann & Mayr 1986] are oriented towards a life-cycle model and a project organisation with phases for action. These approaches put most emphasis on the early stages of the life-cycle. However, they tend to neglect the use and operation of an information system, or an organisation, after implementation. This is certainly not the appropriate way of viewing an information system when one is concerned with its continuing performance over a long period of operation. Generally, a system is built within several weeks or months and used for many years.

The assumptions made about an information system and its environment during its implementation cannot be considered valid throughout its operation. For instance, the throughput can vary with the season or with market trends; new products may emerge, and the size of the system may change. New technology is emerging at a rapid rate, causing changes to the way systems are being built. Communication and networking are becoming more important, causing systems to grow dynamically and to become far more complex.

Hence, for the design and operation of a distributed system a new management method should be followed. This calls for design and management approaches that explicitly surrender the assumptions underlying a phase-oriented approach [Floyd 1981]. Comandos thus adopts an evolutionary approach to system management.

Figure 1

The adaptive framework for the management of Comandos systems, described in figure 1, can be mapped onto three generic types of activities. *Observation activities* are concerned with collecting information while the system is running. *Decision activities* support the actual design, configuration and security management decisions. They are carried out by a human designer or manager using an interactive administration or management tool specific to the task. *Control activities* realise a design or configuration decision, i.e. implement a change. These activities are supported in Comandos by a set of co-operating management tools (see section 2.2).

1.2 Programming with the Comandos Model

To both facilitate the use of the Comandos model and to encourage its adoption, the model is provided to application developers through one or more programming languages. The uniformity of the Comandos model results in programming languages in which a uniform treatment of both transient and persistent data, and of both local and remote services are all potentially available.

This strategy differs from more classical approaches in which tight coupling of a programming language with support for persistence and distribution is not supported. Typically, in the classical approach, a set of languages is used for building programs where one language is used to describe the computation to be carried out, a different language is used to define storage types, and a further language is used for interface definition and communication.

While the Comandos model describes the support provided by the platform in an abstract way, it is important to realise that the interface to the platform used by an application developer is typically provided through one of a number of supported programming languages. The Comandos platform was designed to be independent of the use of any specific programming language for the development of distributed applications and to allow a range of different languages to be supported simultaneously. Indeed, a single application may be composed from parts written in different programming languages. Of course the features of the model are particularly exploitable by object-oriented languages. In any case, the model seen by a programmer may differ from the model implemented by the platform. For example, a particular language may restrict the visibility of certain features of the platform or, alternatively, enhance the functionality of the basic platform by adding extra support in its run-time system.

Two approaches to the provision of language support were followed: supporting existing languages, and providing a new language environment.

The two existing object-oriented languages supported within the project were C++ [Ellis & Stroustrup 1990] and Eiffel [Meyer 1988]. The overall strategy was to enhance each host language with features of the Comandos computational model (mainly concurrency, persistence, distribution and atomicity). Language extensions are supported using appropriate pre-processing which generates necessary supplementary information. In some cases, restrictions are imposed on the use of standard features of the languages due to the distributed nature of the environment. However, in order to be consistent with the definition of each language, as well as to be able to support existing code, the number of such restrictions has been minimised. The paper by Tangney and others in this issue reports on experience with programming parallel distributed applications in the version of C++ supported by the Amadeus/RelaX implementation of the Comandos platform (see section 2.1.1).

The experience drawn from the early phase of Comandos made it clear that it was useful to provide a new language in which all the concepts of the Comandos model are directly reflected. The Comandos object-oriented programming language is known as Guide and is described in the paper by Balter and others in this issue. The viability and usefulness of this new language have already been demonstrated by programming basic system services and a number of distributed applications.

2 Structure of the Comandos Platform

The overall structure of the Comandos platform is presented in figure 2. Essentially, the platform consists of two main components:

- The *Comandos system*, which is itself structured in two layers: the *virtual machine*, which is independent of any supported programming language, and a set of one or more *language-specific run-time systems.*

- A set of *application services and management tools*, which extend the functionality of the basic Comandos system but are built as normal applications.

Figure 2

A language-specific run-time system implements a specific extension of the basic Comandos virtual machine for a particular programming language. A given Comandos system provides at least one language-specific run-time in order to support objects programmed in the corresponding language; several language-specific run-times can coexist in the same system, thus allowing several languages to be used simultaneously.

2.1 The Comandos Virtual Machine

The Comandos virtual machine (see figure 2) provides the basic mechanisms which are necessary for an object-oriented distributed system supporting multiple language environments. This includes transparent access to distributed and persistent objects; the control of distributed computations; transaction management; sharing; and low-level security mechanisms.

The virtual machine provides a unified framework for the management of objects, which supports the viewpoints of both databases and general purpose programming languages. Programming languages frequently deal with transient entities – objects whose lifetime is limited to the execution of a program – while database languages deal with persistent ones, i.e. objects whose lifetime is independent from that of the programs which use them. The virtual machine supports both viewpoints in a uniform way.

At the upper level of the virtual machine is the *generic run-time* (GRT) (see figure 2) which provides a language independent layer implementing distributed object invocation. The GRT is itself provided above the *kernel* layer which includes those components of the virtual machine which must be implemented in a protected way, and which interfaces directly with the underlying host environment.

The virtual machine consists of a number of functional components, which should be present in every Comandos implementation. These components are as follows.

- *Virtual Object Memory.* This is responsible for implementing transparent access to distributed persistent objects and for all aspects of object management including object creation, low-level object naming, object location, remote invocation and the mapping and unmapping of objects to and from secondary storage.
- *Execution Sub-system.* This implements distributed concurrent processing and is responsible for job and activity creation and management, load balancing and low-level synchronisation mechanisms.
- *Storage Sub-system.* This provides distributed persistent storage.
- *Transaction Sub-system.* This implements the mechanisms necessary to support atomic objects and the transaction model.
- *Protection Sub-system.* This provides low-level security services including authorisation, secure transmission and generation of audit data.
- *Communication Sub-system.* This provides network communication services.

The Comandos *Virtual Machine Interface* (VMI) (i.e. the interface between the GRT and the various language-specific run-times (see figure 2)) is intended to be used by compiler builders and basic service implementors. It provides a set of primitives which allow the interaction between application objects and the virtual machine.

The VMI is the uniform view presented by the Comandos virtual machine to each of the various supported languages. As different languages have different calling semantics, a language-specific run-time must adapt the GRT primitives to the language-specific format. Moreover, as most of these primitives are based on manipulation of objects whose format and model differ in each of the different languages, each language-specific run-time must also hide these language dependencies from the GRT.

To provide this flexibility and to make a minimum number of impositions on any language, Comandos adopted a general model in which the language may make calls to the GRT which depend on language-specific information. To handle such a call the GRT makes heavy use of up-calls, where an up-call is a call from a lower level to a higher one, to obtain the necessary information from the language-specific run-time.

This two-way interface between a language-specific run-time and the GRT allows objects from heterogeneous languages to be handled uniformly by the GRT.

2.1.1 Implementations of the Virtual Machine

The definition of the virtual machine is independent of any underlying system, and can be mapped onto various host environments. Two approaches have been used to implement prototypes of the Comandos virtual machine.

- The first approach consisted of implementing the system as a guest layer on top of UNIX. One such implementation, Amadeus/RelaX, was designated as the *reference platform* for the project. Therefore it was the basis for the integration of the numerous system components, application services and management tools developed throughout the project. This implementation is detailed in [Cahill *et al* 1993]. The other major UNIX-based implementation of the virtual machine, IK, is described in [Sousa *et al* 1993].
- The other approach followed was to implement the virtual machine directly on top of a micro-kernel. The motivation for this approach stemmed from the belief that the micro-kernel technology would be better able to support the Comandos abstractions, especially as far as distributed shared objects and protection were concerned. Two prototypes have been implemented using micro-kernel technology: one, Chorus/COOL v2., is hosted on top of the Chorus micro-kernel while another, Guide-2, runs on top of OSF/1-MK. These implementations are described and the resulting conclusions about the use of micro-kernel technology to support the Comandos platform are reported in [Lea *et al* 1993] and [Balter *et al* 1993] respectively.

Each of these prototypes implements a different (if not distinct) subset of the VMI. The existence of several prototype implementations results from a deliberate strategy of investigating different approaches and techniques in the implementation of the core functionality of the virtual machine. In addition each of the implementations focused on different areas. For example, the focus of the Amadeus/RelaX platform was on multiple language support; on support for atomic objects and transactions; and on low level mechanisms to support associative access to objects. In contrast the focus of the IK implementation was on the support of orthogonal persistence in a distributed UNIX environment; on the exploration of on-line replacement of classes as a mechanism to support dynamic configuration and debugging of applications; and on the combination of clustering and naming mechanisms in order to achieve better organisation and structuring of information in the persistent store. It should also be noted that full interworking of the various prototypes was not possible within the timescale and of the project.

2.2 Application Services and Management Tools

The services and tools provided by the platform fall broadly into two categories: application services which are used in the development of distributed applications or which are required at run-time by a range of distributed applications, and management tools which are provided to assist in the management and administration of the system. Application services provided within the project include the following.

- *Development tools* to aid application designers in the design, construction and debugging of distributed applications. These development facilities include compilation tools, a distributed debugger and tools for the development of user interfaces.
- A *Type Manager* (TpM) which can be used by the languages available on a Comandos platform as a repository for type information. During the process of application development or configuration, types can be created and registered in the TpM. Type information may be read for the purpose of semantic checking either at build time or at run time. The TpM's clients are thus compilers, pre-processors, language run-time systems and query processors.
- An *Object Data Management Service* (ODMS) for the management of and associative access to collections of objects [Glasgow 1992]. The ODMS is the component of the platform that performs database-like functions. The ODMS basically implements the BROOM data model facilities (see section 1.1.6). To achieve this goal the ODMS provides various forms of representation of collections, built-in operations on the pre-defined bulk data types of BROOM, and implements mechanisms for the efficient retrieval of objects from collections (e.g. indexing, part-of and inheritance hierarchy scanning, etc.) and optimisation of queries on collections.

The management tools provided include:

- A *Distributed Directory Service* (DDS) for symbolic object naming. The DDS is an implementation of the ISO/CCITT X.500-IS9594 standards integrated into the Comandos platform.
- A distributed *System Observation Facility* (SOF) which supports the observation activities of the adaptive framework for system management described in section 1.1.8. Information collected by the SOF is passed to high-level management tools for processing (see below).
- A distributed *System Control Facility* (SCF) which supports the control activities of the adaptive framework for system management. Administration and reconfiguration decisions stemming from high-level management tools can be implemented using the SCF.
- A *system configuration and administration tool* which is bound to the SOF and SCF. System administrators are provided with graphical output showing statistics concerning system behaviour and can initiate configuration changes.
- *Security tools* for risk management and analysis of audit data generated by the virtual machine. Risks with respect to the integrity, availability, and confidentiality of information can be estimated on the basis of data provided by system observation and auditing as well as by simulation of system failures.
- A *tool for the design of a distributed office system.* Organisational designers are supported in the logical design of business processes and of the distributed information system used to execute these processes.

Several of these tools are discussed further in the paper by Kerber and others in this issue.

In addition, existing tools can also be used, where appropriate, in a given Comandos environment. If the tool exists in a UNIX environment, then recoding of the tool is not required because of the coexistence of the Comandos and UNIX environments.

3 Introduction to the Papers

As will be appreciated, Comandos was a large and ambitious project covering a number of different research topics in areas including (distributed) operating systems, programming language design, fault tolerance and data management. The breadth of the project is reflected in the collection of papers appearing in this issue which cover a cross section of the work carried out in the project.

The paper by Tangney and others takes an application programmers view of one implementation of the platform and reports on experience – both positive and negative – with programming parallel distributed applications in the version of C++ supported by Amadeus/RelaX (see section 2.1.1). The paper presents a list of requirements on a platform supporting such applications based on the experience gained – not all of which are supported by the existing implementation.

The Guide object-oriented programming language is described in the paper by Balter and others in which the authors outline the main features of the language including separation between types and classes, the approach to sub-typing, the synchronisation mechanism for shared objects and the exception handling mechanism. In addition, the authors report on their experiences in using the Guide language for the construction of distributed applications.

The Comandos approach to distributed systems management as well as some of the available tools are discussed in the paper by Kerber and others. In particular the authors discuss the adaptive management approach and its integration into the Comandos model. The system observation and control facilities integrated with the platform are described as well as the tools for distributed information system design, risk management, and user and host administration.

The paper by Sousa and others discusses the implementation of persistence in IK (see section 2.1.1) and discusses the interaction between clustering, object naming and persistence. In addition the authors report on their experience with persistence in a number of applications.

The paper by Taylor and others discusses the design of the two major interfaces concerned with transaction management in Amadeus/RelaX: the (subset of) the VMI concerned with support for atomic objects and transactions, and the Transaction Manager interface. The authors discuss the separation between generic and language-specific aspects of atomic object management and describe a Transaction Manager interface suitable for the requirements of object-oriented systems. In addition, the authors show how an existing XA-compliant resource manager – the Informix relational database system - has been integrated with Amadeus/RelaX.

4 Summary and Conclusions

The Comandos project has integrated operating system, programming language and database technologies in order to design an integrated platform for the construction of object-oriented distributed applications. Along the way Comandos has contributed to the state of the art in (distributed) operating systems, programming languages, data management, and security and administration tools. This claim is supported by the large number of theses and published papers originating from the project.

More importantly, the project has provided prototype implementations of all of the components of the platform – many of which are now available in the public domain – providing the opportunity for real experience to be gained with the technologies developed within the project. Moreover, by adopting a strategy whereby different approaches were taken to the implementation of the key components, the project has provided an opportunity for these different approaches to be compared and the trade-offs between them to be understood in practice. The detailed results of this work are documented in [Cahill *et al* 1993], in the other papers on Comandos in this issue and in the many other publications originating from the project. [Cahill *et al* 1993] contains a full list of these publications and also of the available software components (as of the time of its publication).

To conclude, some of the specific achievements of the project are noted:

- the definition of a conceptual model of, and an architecture for, an integrated application support environment supporting the construction of distributed, persistent object-oriented applications;
- a demonstration that the construction of such an environment is feasible both on UNIX systems and using micro-kernel technology;
- the implementation of a generic run-time system which facilitates the use of existing and new object-oriented languages, for the development of distributed and persistent applications, without requiring each language to adopt a common object model or invocation mechanism;
- the implementation of a new language specifically aimed at the construction of distributed applications involving shared persistent data;
- the definition and implementation of a novel object-oriented data model including binary relations as collections;
- the definition of a canonical type model suitable for capturing the type models of object-oriented programming languages and the implementation of a type manager to support cross-language invocation.
- the implementation and integration of a collection of sophisticated tools for distributed application development and system administration.

Currently no vendor independent infrastructure incorporating all of the features of the Comandos platforms exists in the marketplace. The availability of such an integrated platform, operating in an environment of heterogeneous machines, would be a significant advance over current practice and offer significant advantages to application programmers and system administrators alike.

Acknowledgements

The authors gratefully acknowledge the efforts of all those who contributed to the Comandos project during its seven years.

References

Almes, G., Black, A., Lazowska, E. and Noe, J. (1985) The Eden system: A technical review. *IEEE Transactions on Software Engineering*, SE-11(1), 43–58.

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

Balter, R., Chevalier, P.Y., Freyssinet, A., Hagimont, D., Lacourte, S. and Rousset de Pina, X. (1993) Is the microkernel technology well suited for the support of object-oriented operating systems: the Guide experience. In *Proceedings of the Symposium on Microkernels and Other Kernel Architectures*, 1–11, USENIX Association.

Bernstein, P., Hadzilacos, V. and Goodman, N. (1987) *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA.

Cahill, V., Balter, R., Harris, N. and Rousset de Pina, X. (Eds.) (1993) *The Comandos Distributed Application Platform.* Springer-Verlag, Berlin.

Dasgupta, P., LeBlanc, R. and Appelba, W. (1988) The Clouds distributed operating system. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, 2–9, San-Jose, CA.

Ellis, M.A. and Stroustrup, B. (1990) *The Annotated C++ Reference Manual.* Addison-Wesley, Reading, MA.

Floyd, C. (1981) A process-oriented approach to software development. In *Proceedings of the 6th ACM European Regional Conference*, London, UK., 285–294, ACM, New York.

Glasgow Comandos ODMS Group. (1992) The Comandos ODMS Portable Toolkit. Technical Report DB-92-4, Department of Computing Science, University of Glasgow.

Gray, J. and Reuter, A. (1992) *Transaction Processing Systems: Concepts and Techniques.* Morgan Kaufman Publishers.

Grochla, E. (1982) *Grundlagen der Organisatorischen Gestaltung.* C.E. Poeschel Verlag.

Harper, D.J., Norrie, M.C. and Walker, A.D.M. (1991) Bulk Types for Data Modelling in Persistent Object Systems. *Journal of Control Systems and Machines*, 9.

Kröger, R., Mock, M., Schumann, R., and Lange, F. (1990) RelaX - an extensible architecture supporting reliable distributed applications. In *9th Symposium on Reliable Distributed Systems*, 156–164, Huntsville, Alabama.

Lea, R., Jacquemot, C. and Pillevesse, E. (1983) COOL: System support for distributed programming. Communications of the ACM, 36(9), 37–46.

Liskov, B. and Scheifler, R. (1983) Guardians and actions: Linguistic support for robust, distributed programs. *ACM Transactions on Programming Languages and Systems*, 5(3), 381–404.

Lockemann, P. and Mayr, H. (1986) Information system design: Techniques and software support. In Kugler, H.-J. (Ed.): *Proceedings of the IFIP* 10th *World Computer Congress*, Dublin, Ireland, 617–634, Elsevier Science Publishers, Amsterdam.

Medina, M. and Moreno, A. (1991) Security Levels Supported by the Comandos Security Architecture. In *Proceedings of the 1991 ESPRIT Conference*, Brussels, Belgium, 421–426, Commission of the European Communities, Luxembourg.

Meyer, B. (1988) *Object Oriented Software Construction.* Prentice Hall.

Mullender, S. (1985) *Principles of Distributed Operating System Design.* PhD thesis, Mathematisch Centrum, Vrije Univeriseit.

Popek, G. and Walker, B. (Eds.) (1985) *The LOCUS Distributed System Architecture.* MIT Press, Cambridge, MA.

Schumann, R., Kroeger, R., Mock, M. and Nett, E. (1989) Recovery management in the RelaX distributed transaction layer. In *Proceedings of the* 8th *Symposium on Reliable Distributed Systems*, Seattle, Washington, 21–28, IEEE, Los Alamitos.

Shapiro, M., Gourhant, Y., Habert, S., Mosseri, L., Ruffin, M. and Valot, C. (1989) SOS: An object-oriented operating system – Assessment and perspectives. *Computing Systems*, 2(4), 287–338.

Sousa, P., Sequeira, M., Zúquete, A., Ferreira, P., Lopez, C., Pereira, J., Guedes, P. and Alves Marques, J. (1993) Distribution and persistence in the IK platform: Overview and evaluation. *Computing Systems*, 6(4), 391–424.

Steiner, J., Neumann, C., and Schiller, J. (1988) Kerberos, an authentication service for open network systems. In *Proceedings of the USENIX Winter Conference*, Dallas, TX, 191–202, USENIX, Berkeley.

Weiss, W. and Baur, A. (1990) Analysis of Audit and Protocol Data using Methods from Artifical Intelligence. In *Proceedings of the* 13th *National Computer Security Conference*, Washington DC, 109–114, National Institute of Standards and Technology/National Computer Security Center.

X/Open Company Limited (1991) *Distributed Transaction Processing Reference Model: The XA Specification*, Berkshire.

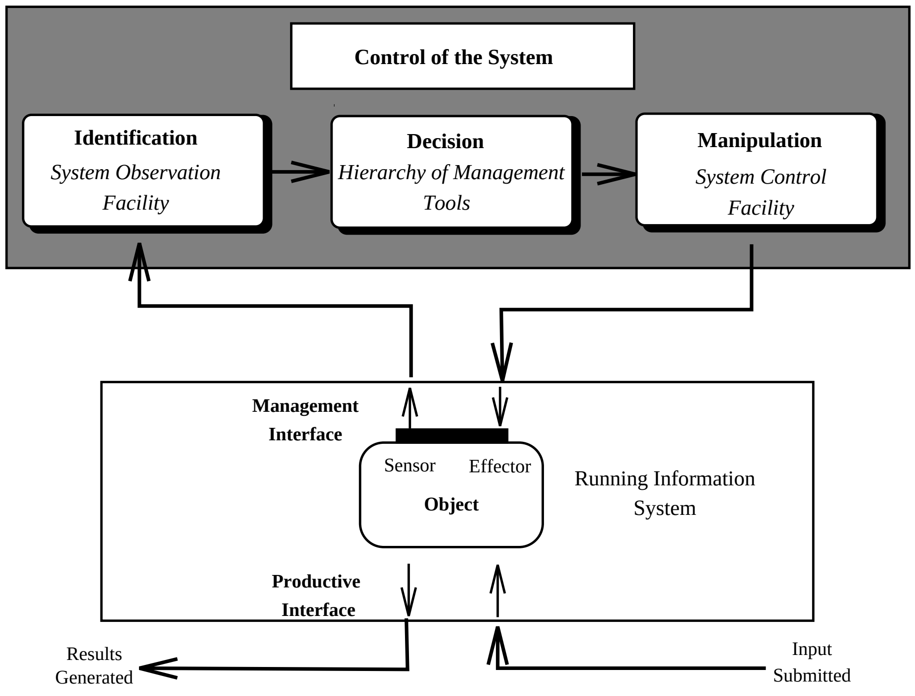
Control of the System
Identification
System Observation Facility
Decision
Hierarchy of Management Tools
Manipulation
System Control Facility
Management Interface
Sensor
Effector
Object
Running Information System
Productive Interface
Results Generated
Input Submitted

Figure 1:

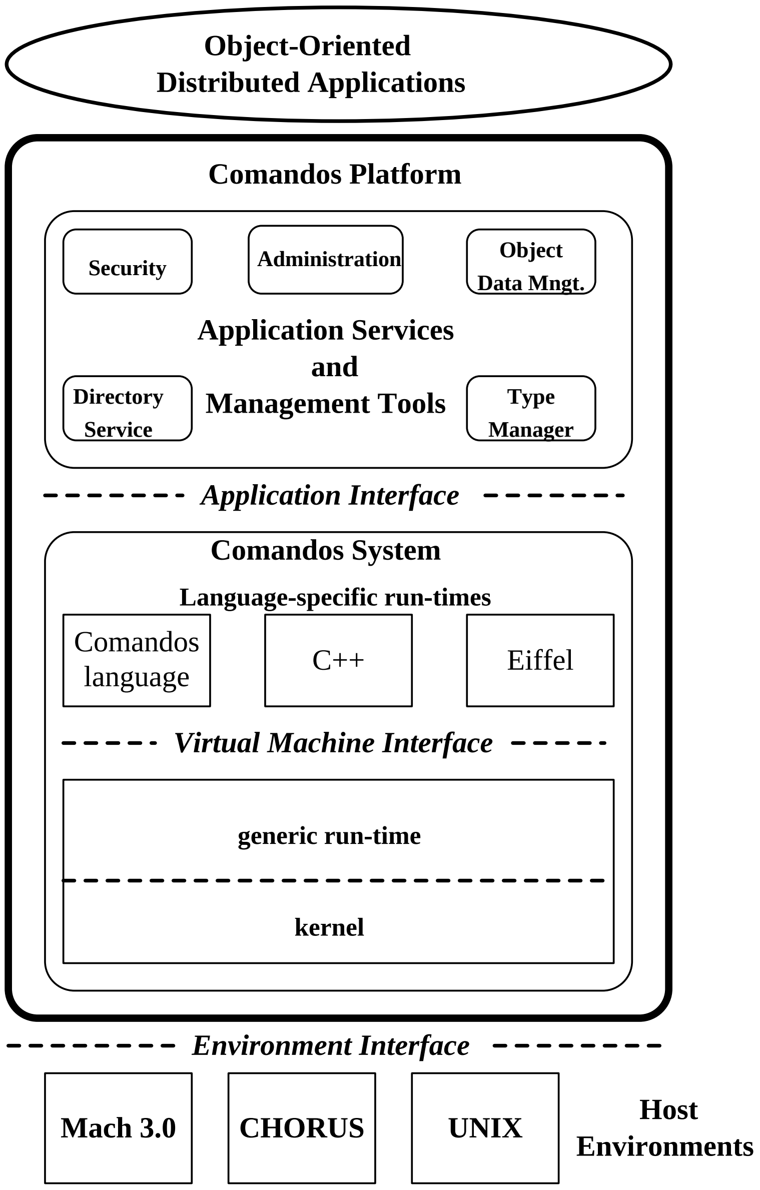
Object-Oriented
Distributed Applications
Comandos Platform
Security
Administration
Object
Data Mngt.
Application Services
and
Management Tools
Directory
Service
Type
Manager
Application Interface
Comandos System
Language-specific run-times
Comandos
language
C++
Eiffel
Virtual Machine Interface
generic run-time
kernel
Environment Interface
Mach 3.0
CHORUS
UNIX
Host
Environments

Figure 2:

List of Figures