***DELLOS* - A Repository Based Environment for Developing Network Centric Applications**

P. Klimathianakis & P. Loucopoulos

**Information Systems Engineering Group, Department of Computation, UMIST
{polivios,pl}@sna.co.umist.ac.uk**

ABSTRACT

The requirements of contemporary enterprises make demands of their support information systems in terms of co-operation, distribution and platform independence. A response of the information systems community to these demands has been the emergence of a new class of applications known as network-centric applications.

The *DELLOS* system provides the basic infrastructure for the development and management of applications which run on any network setting. The system has been implemented using the JAVA language so that the system is usable on any platform and its robust operation over a network is assured. The system also uses repository technology to integrate enterprise data and software components over four levels of abstraction. The behaviour of application components is defined according to a formal model which is itself represented at the highest level of abstraction in the repository.

1.0 Introduction

Many information systems activities demand a co-operative approach involving users of different profiles, background, and culture, and often drawn from different working environments. The globalisation of enterprises, coupled to developments in enterprise integration and the use of information and telecommunications technology to support such developments, have highlighted the importance of the use of tools for assisting users of such systems (designers, software developers, end users etc.) to work in distributed, heterogeneous environments.

Traditional information system development approaches are based on the desktop-centric paradigm offered by the client-server model. More recently however, business needs have encouraged developers to examine different models of working, such as the corporate intranet that is becoming an important development environment for a host of new enterprise applications. The intranet is evolving into a full application development platform, fostering a transformation in application design and implementation. This new class of platform-independent applications, *called network-centric applications*, combines the advantages of central control and administration with the power and flexibility of distributed applications. In contrast to client-server applications that build static connections between a client and its server, network-centric applications accommodate collections of co-operating components and network services. Technologies such as Netscape ONE and Microsoft ActiveX systems are described as open environments for network-centric application development and deployment [3][4][6]. These technologies encourage WEB based applications by offering component interconnectivity through a universal WWW client.

These environments however have been criticised for their poor meta-data management. They lack sufficient and integrated management facilities of the software

components' meta-data such as interface descriptions, class hierarchies, object behaviours. Consequently, the fusion of multiple components from different vendors can be very difficult. As remarked in [5] "As with anything else in the industry, [OLE technology] is great when it works well, but when someone wants to make a change, that causes problems".

The lack of an object model for interchanging of components is considered as a major weakness of current approaches [6].

It is our view, reflected in the design of the *DELLOS* system, that the best way to deal with these problems is to employ a repository for the management of the software components meta-data. We share the view of Bernstein [8] who states that "...by promoting data sharing through common data and information models, and imposing a common set of control services, a repository is the centrepiece of an integrated environment within which a dynamic collection of tools can work together..."

The benefits from adopting such an approach may be summarised as follows:

- the repository provides storage services, so that tool developers need not create tool-specific databases for the objects used by the tools
- the repository allows users to share information simplifying the data exchange
- the repository can hold information which would be subject to common services thus making tools integration an easier proposition.

In our work, integration of tools is achieved through a repository using a model that enables the deployment of applications in a distributed basis. We use the repository not only for data and tool integration, but also for exploiting the notion of applications meta-data. This offers the advantage of being able to create a configuration model for applications taking into account two important facets: (a) the integration of an application with repository data and (b) the evolution of an application and co-operation with other applications within a network-centric environment.

This paper discusses the core elements of the *DELLOS* system. It concentrates on the core facilities that enable system developers to develop network-centric applications. These applications may be related to CASE technology, or to operational systems but, a discussion on specific applications is outside the scope of this paper. In section 2, the paper introduces the topic of network-centric applications and contrasts the approach taken in the *DELLOS* system with other contemporary approaches. Section 3 introduces the main features of the system in terms of its capabilities and in terms of its three fundamental technology threads. These are: its repository; its behavioural model; and its application generation mechanism. These three aspects of the system are described in sections 4, 5 and 6 respectively. Finally, section 7 concludes the paper with a short discussion on the advantages of the *DELLOS* system, its usability, observations from the use of the system on a prototype intranet application development and current and future developments.

2. Network-centric Applications and *DELQS*

It has been argued by practitioners that the need for managing corporate information systems in a distributed manner (either in intranet or internet mode) makes the client-server model inappropriate due to its reliance on proprietary protocols. The response to these needs has been twofold.

Firstly, and historically the earliest response, has been the development of standards such as DCE (Distributed Common Environment) [1] and CORBA (Common Object Request Broker Architecture) [2]. The former offers services for remote procedure calls, domain name service and secure Internet communication; whereas the latter supports a neutral interface definition language and uses the concept of 'broker' to manage co-operation. These standards however, have been regarded by many as being too intricate for integration purposes.

Secondly, and more recently, the network-centric application model has been proposed. This model uses the concept of 'universal client' to hide network-specific details from the application. All network services are abstracted to the extent that any application that is integrated with the universal client is capable of accessing and be accessed by any part of the network. The advantages of this model are:

- *Reduced development cost.* There is no need for developers to learn separate client and server APIs since the same interface can be used in order to gain access to the network
- *Easy migration of code between clients and servers.* This enables the partition of complex applications and permits their growth whilst more components are added on either the client or server side.
- *Reusable applications.* Client applications can act as servers for applications that need to reuse the functionality implemented by the client. Additionally, a WEB of applications can be created where any application acts at the same time as a server for an application and as client for another one.
- *Unified development environments and tools for both server and client.* The common API of the universal client, the common language (HTTP, JAVA etc..), and the hidden network details enables the uniform implementation of all applications.

The universal client model is supported currently by two commercial architectures namely, Netscape ONE and Microsoft ActiveX). Both environments offer similar services to the applications, facilitating their integration over a network, by employing several protocols and architectures.

The *DELQS* approach builds upon the universal client model by using its services. Furthermore, it provides a configuration mechanism for application development based on a formal model for the behaviour of a software component. The motivation for this is the need for ameliorating the problem of complexity in component integration. Usually, component integration relies on script languages [3][4][14] or on integration of programmatic interfaces [1][2]; in both cases derived systems exhibit a high degree of complexity. Each component of an application requires a special treatment and documentation. Hence, the integration of an application becomes very specialised and

only a small number of experienced programmers are able to produce complex applications.

In *DELOS*, this problem is overcome through the use of a common repository for both software and data components. The software components are represented in the repository according to the semantics of a Finite State Machine (FSM) behavioural model which is generic enough to accommodate any software component. The repository itself provides a mechanism for managing the descriptions and we argue, as discussed in detail in subsequent sections, that this mechanism deals more efficiently with complex configurations of network-centric applications than other contemporary approaches.

The *DELOS* network-centric paradigm is an alternative to the universal client model. Instead of arbitrarily connecting components upon a universal client, *DELOS* combines a behavioural model of a component-based application together with the available network infrastructure, building, on-demand, a network-centric application that suits the local user and network requirements. Figure 1 exemplifies the *DELOS* network-centric paradigm.

According to the *DELOS* paradigm there are two aspects that need to be considered by the user of the *DELOS* system in attempting to develop a network-centric application (shown in the diagram as the two 'input' components):

- (i) The component based structure of the application. This is defined using the *DELOS* behavioural model which is itself instantiated and stored in the *DELOS* common repository.
- (ii) The network platform in which a particular application will be deployed. This is defined by a system administrator prior to the application being launched.

Figure 1: The *DELOS* Network-Centric Paradigm

3. Features of *DELOS*

Before turning our attention to the technological advances and characteristics of the *DELOS* system, a short synopsis of its features is provided.

The DELOS system enables easy application customisation and distribution. The *DELOS* system enables component integration in a plug-and-play manner. All software components have a simple interface abstraction with two methods being used, one for 'produced' and one for 'consumed' information. Hence, software components are easily combined into an application by defining the information flow from 'producers' to 'consumers'. The *DELOS* run-time system is responsible for the translation of the produced information to the correct format for the consumer. In addition new components can be added in already defined application configuration or to substitute an old component without any recompilations. The latter is very useful for the distribution of subsequent versions of an application since only the new or changed components need be distributed rather than the entire application.

The DELOS system supports global distributed computing. The system is platform independent. Applications are composed dynamically by pre-fabricated components without the need for any recompilation and can operate anywhere in a network without any physical location restrictions. Any network details are hidden within the *DELOS* environment. *DELOS* is able to derive WEB pages. The main advantage of *DELOS* WEB pages is that one can create not only WEB pages that contain co-operating applets but also connect the WEB pages with enterprise applications.

The DELOS system supports a wide range of communication capabilities. Network-centric applications naturally demand a variety of network facilities. Although current technologies offer technical solutions such as network protocols (HTTP, SMTP, IIOP, etc.) they lack in communication mechanisms that facilitate the creation of complex systems. The *DELOS* system deals with this by offering a number of network protocols with the details hidden from the developer. The network services offered by the system are extensible so that any forthcoming protocols may be accommodated. Moreover, the use of the behavioural model enables *DELOS* to implement a variety of communication mechanisms such as locking protocols, message queuing and transactions.

The DELOS system helps in optimising the application operation. Since *DELOS* applications are composed of software components it is possible for the application developer to create applications that share some of their components. In this way their operational cost and initialisation time may be shared among multiple applications. The *DELOS* system supports dynamic loading of the software components enabling 'on-demand' application configurations and extensions.

The aforementioned features are possible due to the design of the *DELOS* system which is based on three key characteristics:

Repository based system: The repository is the centrepiece of the *DELOS* architecture. It contains and maintains virtually any kind of information regarding an enterprise and its information system *together* with definitions of applications.

Software components described according to a behavioural model: This model describes the behaviour of an application component and the way it is integrated with other components into an application according to a FSM model. The instances of the behavioural model, in the repository, are application descriptions.

Applications generated by a generic mechanism: This mechanism generates an application according to the descriptions stored in the repository by using the software components referenced by the application description.

These three characteristics of the systems are discussed further in sections 4, 5 and 6 respectively.

4. The *DELOS* Repository

The *DELOS* repository is based on the *Semantic Index System* (SIS) [16]. The SIS uses a persistent storage mechanism, based on an Object Oriented semantic network data model [7][9][12] which derives from the *Telos* [13][17] knowledge representation language. The following representation mechanisms are provided:

- *Unbounded multiple instantiation hierarchy:* Each entity in SIS can be declared as an instance of more than one class of the repository. Using SIS one is able to define several levels of abstraction where the entities of level $n+1$ constitute a schema for the entities of the n th level.
- *Multiple strict inheritance (isA):* Entities can be organised in generalisation hierarchies.
- *Multivalued attributes which can have their own attributes:* SIS supports network-like data structures. This is a key difference between SIS and usual Object Oriented systems.

The SIS query system provides search by multiple and recursive conditions, as well as navigation search through the entire network of semantic relations. A query can reference both schema and meta-schema levels. Figure 2 gives an example of the SIS repository.

The *DELOS* repository deals with a large amount of knowledge regarding (a) the software components and the application configuration and (b) the integration of enterprise data with the development applications. The configuration information is stored within the repository and is managed by the repository manager. The repository is able to store and manage all the configuration data much more efficiently than any other system that uses external configuration files. On the other hand, the integration

of the application components with data ensures consistency amongst them and reduces maintenance cost.

A key design decision for the system has been the integration of information about enterprise data with that of applications. This integration is consistent across all levels of abstraction thus enabling different levels of traceability from original business requirements to implementation. This aspect of the *DELOS* repository is shown schematically in the diagram of figure 2.

Figure 2: Overview of the *DELOS* Repository

The repository is partitioned in three models: The *Enterprise model*, the *Information System model*, and the intermediate *dB Service model*. Both enterprise and information system models are tightly integrated and they extend over four levels of abstraction, the Meta-meta level (M2), the Meta level (M1), the Simple class level (S) and the Token level (T). Levels M1 and M2 refer to meta-data, independent from any specific enterprise. Levels S and T correspond to specific enterprise data. Here is a short description of each of the three level:

The *Enterprise model* is concerned with descriptions of an enterprise. We use this term collectively to refer to all aspects of the modelled universe of discourse including conceptual schemata pertaining to the information system supporting the enterprise; but these descriptions are at the user level (as opposed to the descriptions of the information system model which concentrates exclusively on software components). The M2 level refers to generic descriptions of a way of working. The M1 level refers to descriptions of a particular approach which conforms to the M2 way of working definitions. In essence M1 descriptions constitute descriptions defined by a method engineer. On the basis of such a description, one is able to define (and store in the repository) descriptions of particular enterprises at level S. Finally, level T corresponds to enterprise operational data.

The *Information System model* is used for the software components' description and the application configuration and execution. It has four layers of abstraction. The M2 level accommodates the behavioural model that describes the way of working of a particular software component. The M1 level contains the descriptions of the implemented software components according to the behavioural model. In fact the M1 level is a library of software components. This library may be used to create a specific

enterprise information system. The applications of this system would be composed of instances of the software components descriptions and will be stored in the S level. The T level is used by the system during the application execution for storing runtime data.

The *db service* model is the interface between the enterprise data and the information system. It is responsible for maintaining the appropriate links to both enterprise and information systems models, ensuring their consistency. The db services are used by the applications in order to query or update the enterprise data. The main advantage of this architecture is that the enterprise data and the information system models evolve independently without any complications, since the intermediate level remains unchanged. The independence of the software components that constitute the application from the enterprise data simplifies their integration and increases their reusability.

The significant characteristic of the coexistence of the information system model and components with the enterprise models and data. This approach facilitates a smooth and consistent transition among the enterprise concepts vertically (from concepts to data) or horizontally (from methods and requirements to implementations).

5. The *DELOS* Behaviour model

Traditionally, descriptions of software components (SC) have been carried out in terms of static analysers. Although such descriptions contain rich information about a component's interface they do not contain any information about the way of using the particular component. Usually, this information accompanies the description in a textual form.

In our view, the description of the way that a SC behaves is the most valuable piece of information in order to integrate it with another SC. Hence, we need a model to (a) describe the way of working of the software component and (b) use this description in the integration process.

We use a Finite State Machine (FSM) model where an FSM is defined as a triple (K, Σ, s) and a transition function δ . The Information System model discussed in section 4 is underpinned by this FSM.

K is a finite set of states. Although the concept of 'state' does not exist in procedural languages (JAVA, C), developers often design their programs thinking of states. The use of semaphores is one attempt to restrict the access to the program elements. In object oriented design many times the method calls are forced to be in a specific order, otherwise error messages and exceptions are produced. The introduction of the concept of the state is *DELOS* attempts to incorporate the way of operating of a particular SC. The definition of a finite set of states is a task independent from the implementation of the particular SC. The most general case is that every SC has at least one state, the initial state. By defining more states we can indicate changes in the behaviour of the SC. The term "behaviour of the SC" describes the ability to accept or reject a method call at a specific time.

s in K is the initial state. According to the definition of the set K there is always at least the initial state. We call this state s .

Σ is an alphabet. The symbols of the alphabet are those that fire transitions. However, the Σ corresponds to the actual methods of a SC.

δ is the transition function. It is defined as $\mu = \delta(\lambda, \alpha)$, where μ, λ are in K and α is in Σ .

In order to describe the production of the symbols we adopt the metaphor of 'producer/consumer'. We extend the FSM model in a way that each SC within a state is able to produce a subset of Σ . These symbols correspond to the consumed symbols of another FSM, so that the first one is the producer of information and the second the consumer. The combined FSMs simulate the way of working of the application. We can extend or change the functionality of a particular application just by extending or changing the participated FSMs. This is one of the key points of the *DELOS* approach, since it permits fast and flexible re-configuration of applications without needing to recompile anything.

The proposed model that describes the way of working of a software component is shown in the diagram of figure 3. The graphical notation corresponds to the TELOS representation and it is represented at the M2 level of abstraction (see section 4).

Figure 3: The Application Component FSM model

To demonstrate the use of the model consider, as an example, the case of developing a drawing application in a network based co-operative environment. This application consists of two SCs, the graphical display window and the control panel. We limit this example, for reasons of simplicity, to two nodes in a network, working co-operatively.

Graphical Display Window: This SC is a graphical window that can display simple shapes. It consumes two types of event:

- a pair of co-ordinates representing the upper-left and lower-right point (C)
- the type of the shape to be designed (S)

Whenever the graphical display window receives a pair of co-ordinates it designs the appropriate shape using those co-ordinates. The type of the shape is changed whenever this SC receives a new shape type.

Control Panel: This SC is used by users in order to insert their preferred co-ordinates and shapes to be designed on the graphical display window.

Although the implementation of the graphical display window is unique, we can use more than one FSM to represent a variety of behaviours for the particular module. This has the advantage of avoiding re-implementation of code for different applications. This in effect promotes the concept of re-use.

In figure 4 three FSMs are given for the graphical display window and one for the control panel.

Figure 4: SC Description and Application Configuration

The details of the four FSMs, A, B, C and D shown in the diagram of figure 5 are as follows:

A: This is a basic FSM that has only one initial state. This GDW can accept C,S events at any time.   B: A two state FSM. This GDW always require the definition of shape S, before design a shape with co-ordinates C.   C: A two state FSM with the second to be locking state. Whenever this GDW receives an S event changes to state K2, and remains there until a second S event appears.   D: This is a simple FSM for CP: This CP is always able to produce S and C events.

In order to define an application configuration (a) the appropriate FSM models (D and C in our example) are instantiated and (b) the suitable correspondences among the produced and consumed events are established.

The repository representation mechanisms support the specialisation of the defined FSM through inheritance (isA relation) or the extension of the FSM through the

instantiation hierarchy (in relation). In our particular example FSMs B and C are subclasses of FSM A. They are specialisations of the basic FSM A with additional features, such as conditions and locking states. Moreover, The graphical display window FSM that is used in the application is an instance of FSM C. This is an extension of FSM C as it supports two locking states instead of one.

6. The *DELOS* Application Generation Mechanism

The *DELOS* application generation mechanism incorporates the latest developments in object oriented and network technology supported by JAVA [15] which is the implementation language of *DELOS*. The mechanism is able to compose applications on demand according to application specifications that are stored in the *DELOS* repository. It uses the JAVA dynamic class loader in order to generate instances of the software components providing them with their own thread of control. Specialised software modules co-ordinate their operation according to the application description. The diagram of figure 5 gives an overview of the architecture of the *DELOS* application generation mechanism.

Figure 5: The *DELOS* application generation mechanism

A brief description of the application generation mechanism is as follows.

Software components: These components are the building blocks for any application construction. *DELOS* uses components implemented in JAVA such as applets, JAVA classes, and JAVA Beans. Additionally, it uses C/C++ objects wrapped with a JAVA

interface. In a future version it will support Microsoft ActiveX and Netscape ONE components.

JAVA Internet classes: JAVA has an extensive library of classes that implement several TCP/IP protocols such as HTTP and FTP and offers services such as Remote Method Invocation (RMI) and Object Serialisation. *DELOS* takes advantage of these classes ensuring the operation of the system over a wide range of platforms.

DELOS system: *DELOS* is made of the *DELOS* repository (already discussed in section 4), the *DELOS Kernel* that represents the virtual space in which applications are generated and executed, the *application controller* which is responsible for the operation of a particular application, and the *DELOS wrapper* which provides a SC with a uniform interface enabling their integration. The diagram of figure 6 gives a schematic representation.

Figure 6: The *DELOS* System

DELOS Kernel: This is responsible for the operation of the *DELOS* installations. It offers a set of basic services such as downloading a SC from the network or the local file system, creating an application controller, initialising it with the configuration data, associating the downloaded SC with the appropriate application controller and allocating the SC in the available network nodes.

Application Controller: This is the main component of an application. It constitutes the implementation of the application's FSM configuration. It is responsible for dispatching an event when the particular SC is ready to accept it, translating events to a different format if it is necessary, and replicating them if there is more than one receiver for the same event.

DELOS Wrapper: This is a JAVA class providing a SC with an interface abstraction for consuming and producing events. A SC inherits that interface by being defined as a subclass of the *DELOS* wrapper.

Both the application controller and *DELOS* wrapper are implemented as JAVA *unicast remote servers*[15]. Therefore, they are both capable of exchanging messages over a

network by using the JAVA RMI mechanism. This enables the distribution of the various SC virtually anywhere in the network.

7. Conclusions

The *DELOS* system has been entirely developed in the JAVA language. Its usability and portability therefore are greatly enhanced and since the system makes use of the JAVA RMI mechanism it can be assured that any *DELOS* software component can be effectively executed at any node in an intra- or internet setting. But, *DELOS* is more than just a JAVA supporting tool. It is rather a new paradigm for the development of network-centric applications. In this paradigm there is a clear distinction between an application component and the network nature of a network-centric application. This provides the basis for the co-operation of software component engineers, application designers and network administrators working towards the successful deployment of the network-centric model.

The way of developing applications in *DELOS* is akin to 'plug and play' approach (a kind of construction using application components as one would use components of Lego!). The central management and deployment of applications enables system administrators to create local application configurations that are used by the *DELOS* application generator to produce and automatically deliver the network-centric application. Furthermore, *DELOS* enables the exposure of software components to WWW pages, providing a shortcut for the integration of enterprise information systems with the WEB. These facilities have already been proved to work through the use of different prototypes developed to test the network-centric nature of the system.

Currently, work is under way to develop different classes of application exploiting the facilities of the *DELOS* system. These developments include, amongst others, an application to provide access to all ODBC compatible commercial databases, a generic editor to be used as part of a meta-CASE tool, an application that takes as input the results of a database query and produces SGML representations, and tools which can visualise SGML representations in 2D as well as 3D.

It is our belief that the globalisation of enterprises coupled to recent developments in network systems will increasingly place demands on information systems engineers, for technologies that support co-operative working in distributed way using heterogeneous systems. The *DELOS* system represents an effort in meeting such a demand.

References

- [1] DCE Overview, <http://www.osf.org/comm/lit/OSF-DCE-PD-1090-4.html>, Copyright 1992. Open Software Foundation, Inc. All Rights Reserved.
- [2] CORBA 2.0 Specification <http://www.omg.org/corba2/corb2prf.htm>

- [3] THE NETSCAPE ONE DEVELOPMENT ENVIRONMENT VISION AND PRODUCT ROADMAP
[http://home.netscape.com/comprod/one/white_paper.html]
- [4] Microsofts DCOM-LAN LAN
[<http://www.lantimes.com/lantimes/96may/605b001d.html>].
- [5] Emperor's new Clothes
[<http://www.lantimes.com/lantimes/96apr/604c041a.html>].
- [6] Microsoft vs. Netscape - [<http://www.lantimes.com/96mar/603a001d.html>].
- [7] G. Attardi and M. Simi. "Completeness and Consistency of OMEGA, A Logic for Knowledge Representation", In Proceedings of International Joint Conference on Artificial Intelligence, Vancouver, 1991.
- [8] P. A. Bernstein and U. Dayal. "An Overview of Repository Technology", In Proceedings of the 20th VLDB Conference, pages 705-713, Santiago, Chile, 1994.
- [9] M. Brodie, j. Mylopoulos and J. Schmidt, editors. "On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases and Programming Languages", Springer-Verlag, 1984.
- [10] P. Constantopoulos and M. Doerr "Component Classification in the Software Information Base", In O.Nierstrasz and D.Tsichritzis eds., Object-Oriented Software Composition, Prentice-Hall, 1995.
- [11] P. Constantopoulos and M. Doerr. "The Semantic Index System: A brief presentation", Institute of Computer Science Foundation for Research and Technology Hellas, May 1994. [<http://www.ics.forth.gr/proj/isst/Systems/sis/>].
- [12] R. Hull and R. King "Semantic Database Modelling", ACM Computing Surveys, 19(3): 202-260, September 1987
- [13] J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis. "Telos: Representing Knowledge about Information Systems", ACM Transactions on Information Systems, 8(4), October 1990.
- [14] Neuron Data Elements Environment,
[<http://www.neurondata.com/products/ee2.htm>]
- [15] The JAVA Language Environment A White Paper James Gosling Henry McGilton May 1996, http://www.javasoft.com/doc/language_environment/
- [16] The Semantic Index System - SIS (A brief presentation),
[<http://www.ics.forth.gr/proj/isst/Systems/SIS/index.html>]
- [17] J. Mylopoulos, "Conceptual Modeling and Telos", in 'Conceptual Modeling, Databases, and CASE', P. Loucopoulos and R. Zicari (ed.), John Wiley & Sons, 1st, pp. 49-68.