Defence Research and Development Canada
Recherche et développement pour la défense Canada

Introduction to netMiro

Extending Middleware for Robots (Miro) to the .NET Platform

S.L. Bogner
DRDC Suffield

Technical Memorandum
DRDC Suffield TM 2006-187
December 2006

Canada

Introduction to netMiro

Extending Middleware for Robots (Miro) to the .NET Platform

S.L. Bogner
Defence R&D Canada – Suffield

Defence R&D Canada – Suffield

Technical Memorandum
DRDC Suffield TM 2006-187
December 2006

Author

Stephen Bogner, P.Eng.

Approved by

Doug Hanna

Head – Autonomous Intelligent Systems Section

Approved for release by

Dr. P. D'Agostino

Chairman - Document Review Committee - DRDC Suffield

Abstract

Middleware for Robots (Miro) is an Open Source, Object Oriented “robotics-centric” software framework, which exploits the Adaptive Communications Environment (ACE) and the Common Object Request Broker Architecture (CORBA) built on it – The Ace Orb (TAO) - to provide the communications services necessary for distributed computation – a pervasive requirement in robotics.

After conducting a detailed review of available technologies for robotics middleware, Defence R&D Canada (DRDC) Suffield Autonomous Intelligent Systems Section (AISS) decided to exploit Miro within the DRDC program.

A design objective of any middleware technology is to support interoperability across heterogeneous platforms. Miro achieves this through the use of CORBA. Unfortunately, the fact that Miro has been implemented on Linux platforms, using ANSI C++ and Linux toolsets, for a target audience of Linux users, has created a barrier to its acceptance and use by programmers who are more comfortable developing software in the managed C# language using the .NET framework within the Windows environment.

The netMiro project was undertaken to extend the Miro technology to the Windows .NET platform.

This report provides a brief introduction to the netMiro software package.

Résumé

Un intergiciel pour robots (Miro) est un cadre conceptuel de logiciel de source ouverte et centré-objet « robotique-centrique » qui exploite l’Environnement adaptif de transmissions (ACE) et l’Architecture commune de répartition de requêtes d’objets (CORBA) qui y a été construite - Ace Orb (TAO) – pour procurer les services de communication nécessaires à la répartition des calculs – une forte exigence en robotique.

Après avoir effectué une étude détaillée des technologies disponibles pour les intergiciels en robotique, la Section des systèmes intelligents autonomes (SSIA) de R & D pour la défense Canada (Suffield) (RDDC) a décidé d’exploiter Miro dans le programme de RDDC.

Un objectif conceptuel de toute technologie d’intergiciels est de soutenir l’interopérabilité entre les plateformes hétérogènes. Miro réalise ce but au moyen de CORBA. Malheureusement, le fait que Miro a été implémenté sur des plateformes Linux, en utilisant ANSI C++ et des outils Linux, pour un public-cible utilisateur de Linux, a créé une barrière à son acceptation et à son utilisation par des programmeurs qui sont plus à l’aise à mettre au point de logiciels dans le langage géré par C#, utilisant un cadre conceptuel .NET dans un environnement Windows.

Le projet netMiro avait été entrepris pour étendre la technologie Miro à la plateforme Windows .NET.

Ce rapport est une brève introduction au progiciel netMiro.

This page intentionally left blank.

Executive Summary

Middleware for Robots (Miro) is an Open Source, Object Oriented "robotics-centric" framework, that exploits the Adaptive Communications Environment (ACE) and the Common Object Request Broker Architecture (CORBA) component built on it – The Ace Orb (TAO) - to provide the communications services necessary for distributed computation – a pervasive requirement in robotics.

After conducting a detailed review of available technologies for robotics middleware, Defence R&D Canada (DRDC) Suffield Autonomous Intelligent Systems Section (AISS) decided to exploit Miro within the DRDC program. The original branch of the software, developed at the University of Ulm in Germany, is known as ulmMiro; the DRDC branch of extensions is known as drdcMiro.

Unfortunately, the fact that ulmMiro and drdcMiro have been implemented on Linux platforms, using ANSI C++ and Linux toolsets, for a target audience of Linux users, has created a barrier to its acceptance and use by programmers who are more comfortable developing software in the managed C# language using the .NET framework within the Windows environment.

The netMiro project was undertaken to extend the Miro technology to the Windows .NET platform.

netMiro currently supports .Net Remoting style Direct Connections, as well as Object Management Group (OMG) standard CORBA Name Service, Event Service, and Notification Service hosted suppliers and consumers.

The netMiro package makes existing and future ulmMiro and drdcMiro "supplier" objects accessible to .Net "consumers" in the Windows environment, and provides a framework within which "supplier" objects implemented from the Windows environment can be provided to both Windows and Linux "consumers".

Bogner, S.L. 2006. Introduction to netMiro: Extending Middleware for Robots (Miro) to the .NET Platform. DRDC Suffield TM 2006-187. Defence R&D Canada – Suffield.

Sommaire

Un intergiciel pour robots (Miro) est un cadre conceptuel de logiciel de source ouverte et centré-objet « robotique-centrique » qui exploite l'Environnement adaptif de transmissions (ACE) et l'Architecture commune de répartition de requêtes d'objets (CORBA) qui y a été construite - Ace Orb (TAO) – pour procurer les services de communication nécessaires à la répartition des calculs – une forte exigence en robotique.

Après avoir effectué une étude détaillée des technologies disponibles pour les intergiciels en robotique, la Section des systèmes intelligents autonomes (SSIA) de R & D pour la défense Canada (Suffield) (RDDC) a décidé d'exploiter Miro dans le programme de RDDC. La branche d'origine du logiciel, mis au point à l'université de Ulm en Allemagne, est connue sous le nom de ulmMiro; la branche RDDC des extensions est appelée rddcMiro.

Malheureusement, le fait que Miro a été implémenté sur des plateformes Linux, en utilisant ANSI C++ et des outils Linux, pour un public-cible utilisateur de Linux, a créé une barrière à son acceptation et à son utilisation par des programmeurs qui sont plus à l'aise à mettre au point de logiciels dans le langage géré par C#, utilisant un cadre conceptuel .NET dans un environnement Windows.

Le projet netMiro avait été entrepris pour étendre la technologie Miro à la plateforme Windows .NET.

Net.Miro soutient actuellement des connexions directes de style .Net Remoting, ainsi qu'un service de noms, un service d'évènements et un service de notification des fournisseurs et des consommateurs affichés, gérés par Object Management Group (OMG) de standard CORBA.

Le progiciel netMiro rend les objets des futurs « fournisseurs » ulmMiro et rddcMiro accessibles aux « consommateurs » .Net dans l'environnement Windows et fournit un cadre conceptuel dans lequel les objets du « fournisseur » implémentés dans un environnement Windows peut être fourni à la fois aux « consommateurs » Windows et Linux.

Bogner, S.L. 2006. Introduction to netMiro: Extending Middleware for Robots (Miro) to the .NET Platform. DRDC Suffield TM 2006-187. R & D pour la défense Canada – Suffield.

Table of Contents

List of Figures

List of Tables

1 Introduction

This section provides an introduction to the concept of "middleware", and provides a very brief history and overview of the major packages that are combined to create Miro (Figure 1).

1.1 Miro (Middleware for Robots)

Distributed computation is a challenging and ever-present requirement in robotics; the ability to exploit efficient and practical software mechanisms to achieve distributed computing is a fundamental programming capability for any robotics research effort.

From virtually the beginning of Software Science, the need to connect software components or applications across interprocess and machine boundaries has been well known.[1] "Middleware" is the term given to the software layer that mediates between the lower level operating systems and the higher level applications running at each site in a distributed system.

Krakowiak [2] lists the following functions of middleware:

1. Hiding distribution, i.e. the fact that an application is usually made up of many interconnected parts running in distributed locations;

2. Hiding the heterogeneity of the various hardware components, operating systems and communication protocols;

3. Providing uniform, standard, high-level interfaces to the application developers and integrators, so that applications can be easily composed, reused, ported, and made to interoperate;

4. Supplying a set of common services to perform various general purpose functions, in order to avoid duplicating efforts and to facilitate collaboration between applications.

Recently, a consensus (as discussed in Broten et al. [3, 4, 5], for example) has begun to emerge around the idea that the discipline of Robotics could benefit from a specialized middleware having all (or at least some) of the following characteristics:

1. "Robotics Centric" – focus on the practical needs of roboticists and apply to the actual hardware and software technologies being used by working roboticists.

2. "Object Oriented" – embrace the modern object oriented design paradigm, with abstractions rooted in the ontological realities of the robotics domain.

3. "Open Source" – remove any dependencies on proprietary software that is both expensive and difficult (or impossible) to integrate or extend.

[1] The first known reference to "middleware" as "software in the middle of application software and system software" is attributed to d'Agapeyeff and occurred in the final report of the 1968 NATO Software Engineering Conference[1].

4. "Standards Based" – conform to the best practice in modern software engineering, while continuing to support widely accepted and utilized engineering standards (such as CORBA), with the goal of enhancing interoperability across heterogeneous platforms.

5. "Simple to Use" – have a straight forward usage model that could be easily understood and employed successfully by non-specialists, including clear and well documented interfaces and coding templates to facilitate code reuse.

Academics at the University of Ulm (Germany) responded to this consensus[2], beginning in about 1999 and ongoing, by developing a "Middleware for Robots", which they called "Miro". [6]

Miro is an Open Source, Object Oriented "robotics-centric" software framework, that exploits the Adaptive Communications Environment (ACE)[3] and the Object Management Group (OMG)[4] standard Common Object Request Broker Architecture (CORBA)[5] compliant Object Request Broker built on it – The Ace Orb (TAO)[6] - to provide the communications infrastructure and services necessary for distributed computation.

A design objective of any middleware technology is to support interoperability across heterogeneous platforms. Miro does achieve this through the use of CORBA; however in practice the Miro implementations to date have been entirely Linux-centric (in that they have been developed using ANSI C++ on Linux platforms, using Linux based toolsets, for a target audience of Linux programmers).

The netMiro project was undertaken to extend the Miro technology to the C# language on the Windows .NET platform.

1.2 ulmMiro

The original implementation of Miro was produced by developers from the University of Ulm. The software and associated documentation can be found at the Miro Homepage [11]

http://smart.informatick.uni-ulm.de/Miro/index.html

Known as Miro, and maintained as such by the developers at the University of Ulm, Defence R&D Canada (DRDC) has chosen to rename the Ulm branch of the software "ulmMiro" internally, to differentiate it from the DRDC developed branch (with extensions) which is

[2] A few of the other notable initiatives to provide middleware frameworks for robotics include CLARAty from JPL, CARMEN from CMU, ORCA from the Australian Centre for Field Robotics, and MARIE from the University of Sherbrooke.

[3] See http://www.cs.wustl.edu/~schmidt/ACE.html [7].

[4] See http://www.omg.org/ [8].

[5] See http://www.omg.org/gettingstarted/corbafaq.htm [9].

[6] See http://www.cs.wustl.edu/~schmidt/TAO.html [10].

known as “drdcMiro”. The reason for this is to allow DRDC to more easily integrate code updates from the University of Ulm and others, while keeping the DRDC code separate.

ulmMiro builds directly upon ACE/TAO, and has been developed in ANSI C++ for Linux platforms.

1.3 drdcMiro

After conducting a detailed review of available technologies for robotics middleware, DRDC Autonomous Intelligent Systems Section (AISS)[7] decided to exploit Miro within the DRDC program. [3,4,5]

In the course of integrating ulmMiro within the DRDC program, additional drivers and utility programs have been added and these contributions have been packaged within drdcMiro. In general, AISS researchers who use Miro consider themselves to be “users” rather than “developers”, and are very pragmatic in their use of the software.[8]

Like ulmMiro, drdcMiro has also been developed in ANSI C++ for use on Linux platforms.

1.4 netMiro

Unfortunately, the C++ Linux-centric reality of ulmMiro and drdcMiro has created a barrier to the acceptance and use of Miro by programmers who are more comfortable developing software in the Windows environment.

The objective of the netMiro project is to make the Miro technology accessible to programmers developing in the C# language using the .Net frameworks on Windows platforms. netMiro does not use the Miro framework directly, as can be seen by the fact that netMiro can be installed and runs successfully without reference to any of the ulmMiro or drdcMiro code. Instead, netMiro uses OMG CORBA and .Net Remoting technologies to duplicate the essential functionality of Miro, making existing and future ulmMiro and drdcMiro “supplier” objects accessible to .Net “consumers” in the Windows environment, and providing a framework within which “supplier” objects implemented from the Windows environment can be provided to both Windows and Linux “consumers”.

The implementation language for netMiro is C#; however it is directly accessible by any .Net language through the generated library (netMiro.dll).

---

[7] Located at Canadian Forces Base Suffield, near Medicine Hat, Alberta, Canada.

[8] In conversation, most DRDC users – even those who have spent a lot of time working with Miro - consider themselves to be “users, not developers” with respect to Miro, and most also feel that they have a rather shallow understanding overall of the inner workings of ACE/TAO or CORBA (something which may be true of robotics researchers generally, given the extreme complexity of these technologies). When combined with the rather steep learning curve that must be overcome to use a middleware product like Miro effectively, it would be fair to report some reluctance to experiment with alternative ways of using it for fear of disrupting a complex software system that appears to be working acceptably – if mysteriously. Hopefully, netMiro can simplify the use of Miro and reduce the risks associated with such experimentation.

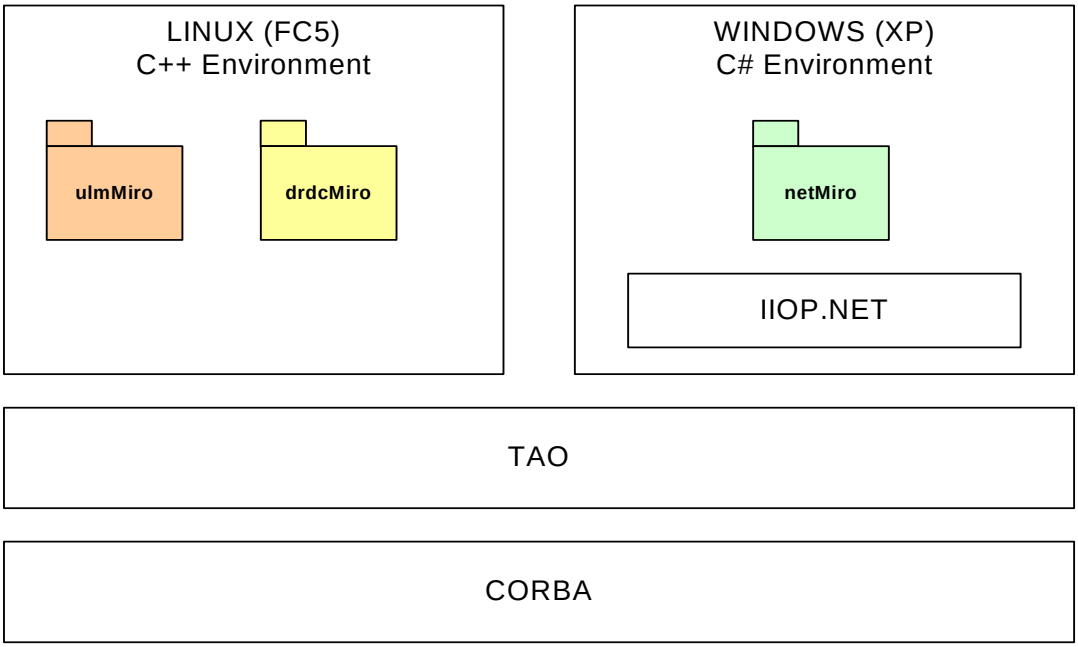

Figure 1 – Middleware for Robots (Miro)

2 netMiro in Context

This section contains background information that will provide necessary context for the discussion about how to use netMiro (Section 3).

2.1 Suppliers and Consumers

2.1.1 Suppliers

A netMiro "Supplier" is any class instantiation (an object) that generates (i.e. supplies) data, including events, for the use of other objects. To write a supplier it is necessary to create (or to have available) an actual "implementation" of the supplier object. However, if the data object that is being passed from the supplier to the consumer is a structure, then both the supplier and the consumer can reference the interface library created from the Interface Definition Language (.idl) file.

The implementation will be written for the environment where the object will run. In almost all practical cases, the supplier object will be connected somehow to the operation of an actual device that generates data of interest, or that processes such data as a consumer, and makes the results of its processing available to downstream consumers.

It must be noted that a great many practical consumer and commercial devices of interest to roboticists are provided with drivers for Windows computers. Hopefully, netMiro will help to

make some of these devices more readily available as suppliers for Miro consumers on both Windows and Linux platforms.

2.1.2 Consumers

A netMiro “Consumer” is any class instantiation (an object) that uses (i.e. consumes) data, including events, that has been generated by another object. To write a consumer, it is only necessary to create (or have available) the “interface” for the consumed object. In many cases, the interface will be provided by .idl files, which can be compiled into interface libraries.

2.2 Communication Channels

When working with suppliers and consumers in a homogeneous environment, it is possible to use IPC, TCP and HTTP channels, such as those provided natively within the .Net framework, or to use a CORBA Internet Inter-Orb Protocol (IIOP) channel. However, when using a CORBA Naming Service, Event Service, or Notification Service it is necessary to use a CORBA supported protocol. In netMiro the IIOP protocol is used (Figure 2).

IIOP.Net is an Open Source OMG compliant .NET channel that supports IIOP protocol communications. The software and documentation for IIOP.Net can be found at the project homepage [12]:

http://iiop-net.sourceforge.net/

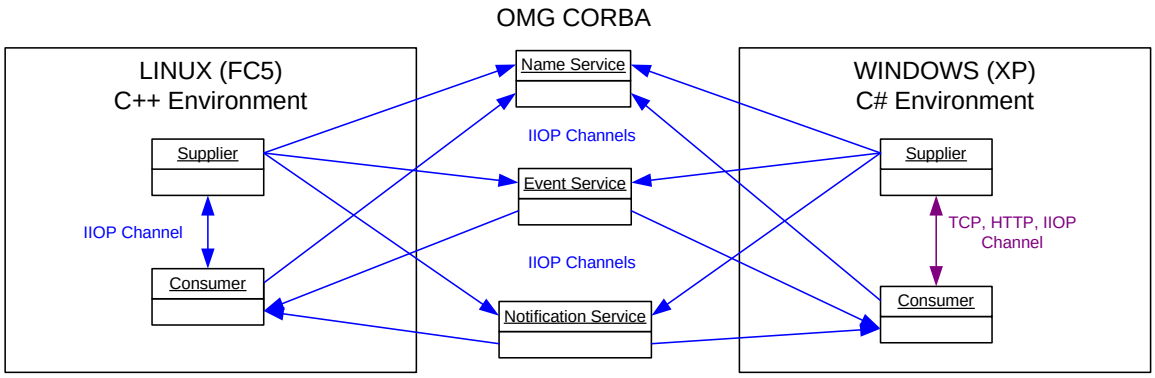

Figure 2 – Communication Channels in netMiro

2.3 Direct Connections

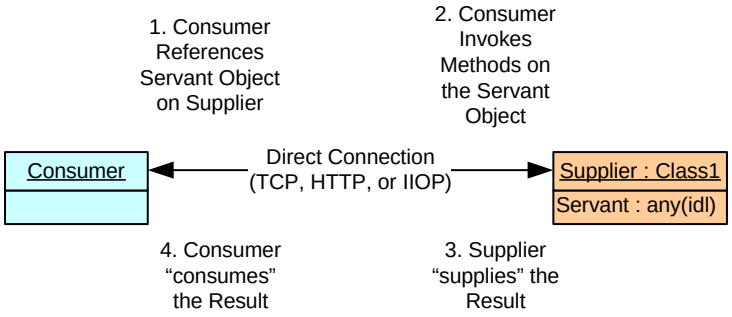

Figure 3 – Communication using a Direct Connection

In netMiro Direct Connections (Figure 3) provide a local reference to a remote "servant" object. In general, the local application will usually be a client/consumer, and the remote application that hosts the servant object will usually be a server/supplier. (Of course, many applications may act as suppliers and consumers simultaneously).

netMiro can connect suppliers and consumers directly using IPC, TCP, HTTP, or IIOP channels. In this case, the local reference provides all of the methods, events, and properties defined for the remote object within its interface. IPC (Inter-Process Communication) is used when both the supplier and consumer are running on the same machine, bypassing the networking layer entirely to deliver significantly better performance.

netMiro supports the full spectrum of synchronous and asynchronous communications capabilities (including callbacks) across the full spectrum of client-created and server-created objects that are available from the .NET framework.

The principal advantages of Direct Connections are:

1. An OMG Naming Service is not required.
2. Channels can be established over IPC, TCP, HTTP, and IIOP.
3. Consumers connect directly to specific well-known suppliers, with every connection being independent of other connections. This provides a robust and maximally distributed network.

The principal disadvantages of Direct Connections are:

1. Consumers must know the Universal Resource Identifier (URI) of suppliers before they are able to connect to them. (This disadvantage is overcome by the use of the Naming Service).
2. Consumers and suppliers must manage their own communications channels. This can be burdensome to suppliers that need to respond to many consumers, and limits the scalability of these systems. (This disadvantage is overcome by the use of the Event Service).

2.4 Naming Service

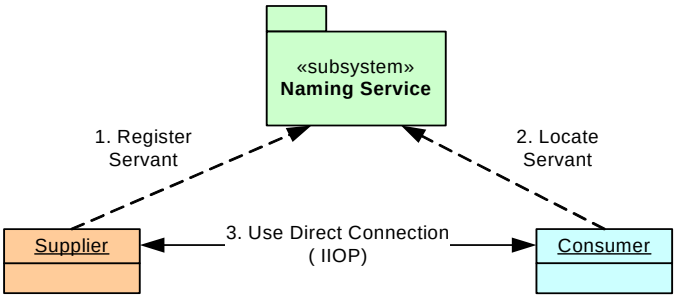

Figure 4 – Communication using a Naming Service

netMiro can connect suppliers and consumers directly through the mediation of an OMG Naming Service on an IIOP channel (Figure 4). In this case also, the local reference provides all of the methods, events, and properties defined in the interface for the remote object, just like a Direct Connection. The specification for the OMG Naming Service can be found at [13]:

http://www.omg.org/technology/documents/formal/naming_service.htm

The Naming Service mechanism provides the advantages of distributed direct connections, with the additional advantage of a central "dating service" to put suppliers and consumers together without having to have a "well known" URI at which to locate the supplier.

Although the Naming Service is a bottleneck that controls the initial establishment of distributed connections, the network that is created is still maximally distributed and therefore robust. If the Naming Service is destroyed, any existing, active connections between the suppliers and their consumers persist.

A Naming Service should be available at the startup of any Miro-based system; therefore the use of the Naming Service ought to be a default usage template for creating an operational supplier and consumer.

2.5 Event Service

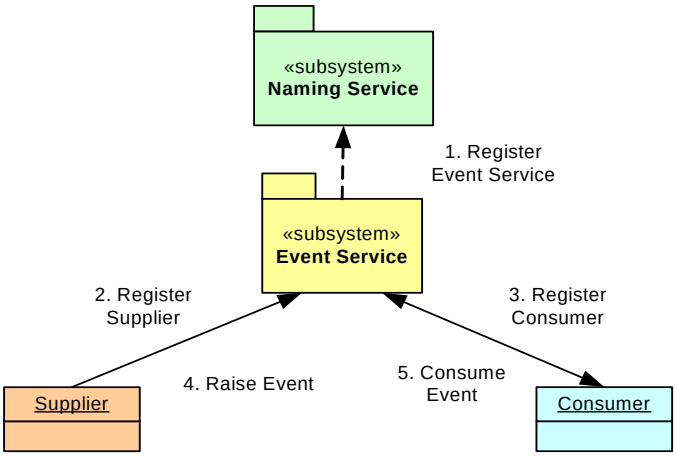

Figure 5 – Communications using the Event Service

netMiro can connect suppliers and consumers through an OMG Event Service, running on an OMG Naming Service using an IIOP channel (Figure 5). In this case, the data that is passed to the consumer from the supplier is limited to a data object passed within the EventChannel.proxyPushConsumer.push() method, which will typically be a custom structure of some type. This custom structure ought to be defined in an appropriate .idl file, so that it is available to both Linux and Windows consumers regardless of the supplier environment. The specification for the OMG Event Service can be found at [14]:

http://www.omg.org/technology/documents/formal/event_service.htm

The Event Service passes all events received from all suppliers connected to it to all of the consumers connected to it. The most logical use for an Event Service is when a single supplier needs to provide the same data to many consumers. The Event Service accepts the single event from the supplier, and sends as many copies of the event as necessary to the connected consumers. This transfers the burden of the duplication task and the communication task to the Event Service, allowing the supplier to concentrate its resources on its own computations. This is a very scaleable service; if too many consumers are connecting to the first event service then a second event service can be instantiated as a consumer on the first, and new connections can be added to it instead.

If the Event Service is destroyed or becomes unreachable, all connections are destroyed with it. However, one ought not to consider the Event Service to be a network bottleneck as such; the architecture is in fact an improvement over having a supplier supporting multiple consumers over direct connections – if the supplier is destroyed or becomes unreachable all of its direct connections would also be destroyed anyway.

The Event Service should be the default usage template for a “broadcasting supplier” where a single supplier is expected to send identical data to many consumers simultaneously.

2.6 Notification Service

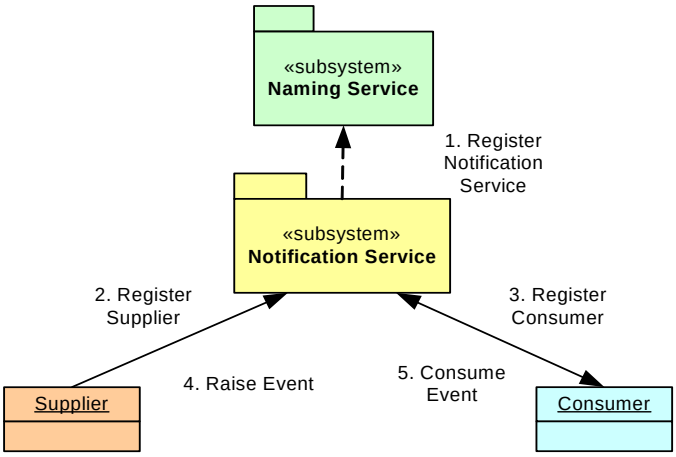

Figure 6 – Communication using the Notification Service

netMiro can connect suppliers and consumers through an OMG Notification Service, running on an OMG Naming Service, using IIOP (Figure 6). In this case, suppliers can specify the collection of event types that they offer, and consumers can subscribe to the subset of these events that they want to receive. The notification service keeps track of these offers and wants, and manages the process of copying and dispatching supplied events to subscribed consumers. The specification of the OMG Notification Service can be found at [15]:

http://www.omg.org/technology/documents/formal/notification_service.htm

Many suppliers and many consumers can be attached to the notification service simultaneously. Many suppliers can be providing some of the same types of events, and any consumer that has subscribed for those types of events will receive all of them, regardless of the source. Similarly, if no consumer is interested in a particular type of event, the notification service will not process those events further when they are received from a supplier.

In addition, a notification service can manage Quality of Service (QoS) properties to control the priority, timeliness, and reliability of the passage of events from suppliers to consumers.

The Notification Service is a significant extension of the Event Service, and it offers important and sophisticated capabilities including a standard for transmitting events in the form of a specific data structure (the StructuredEvent), the ability to publish and subscribe to collections of events, an ability to filter for events meeting specific criteria, a mechanism to dynamically discover the events that are being offered on the Notification Service, and the ability to manage QoS properties. However, these capabilities come at the cost of significantly greater implementation and usage complexity, and many of these sophisticated capabilities are rarely necessary in actual robotic systems.

The most noteworthy disadvantage of the Notification Service is that it can create a significant network bottleneck, particularly if it is used inappropriately or unnecessarily. Architecturally, the Notification Service becomes a central node - as well as a critical single point of failure - in a network of many suppliers and many consumers. The existence of this central node is antithetical to the advantages of distributed computing – the main reason for using Miro in the first place.

One usage scenario where the Notification Service ought to be considered is to support an "aggregator consumer", where many suppliers need to communicate with a single consumer. Examples of this usage include a "Logger Application" that records events from multiple suppliers, or a "Centralized Control Station" that needs to monitor the entire system. However, in general the use of the Notification Service should be undertaken with great caution.

Nevertheless, if the system requirements actually demand the capabilities of the Notification Service, netMiro will support them.

3 Using netMiro

This section will provide guidance about how to set up the development environment so that netMiro (Figure 7) can be successfully compiled and extended. It will then provide a summary of each of the examples that are provided with the netMiro package. Developers are encouraged to use these examples as starting templates for producing their own operational netMiro suppliers and consumers.

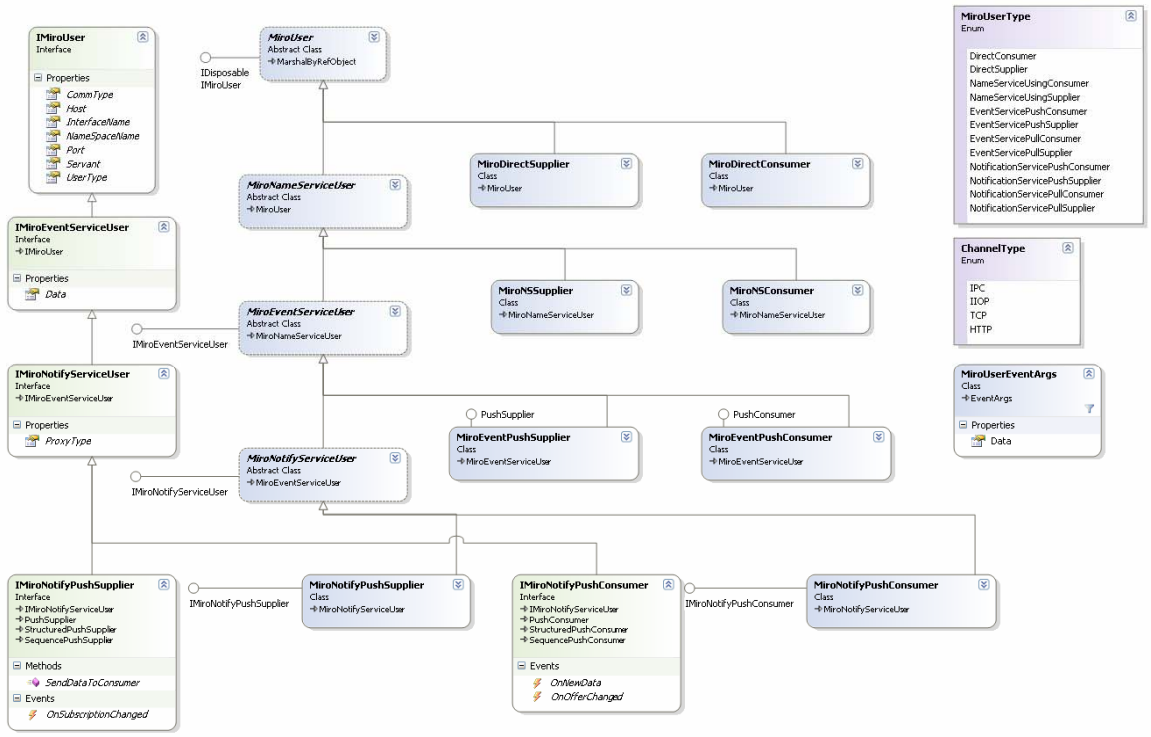

Figure 7 – netMiro Classes, Interfaces, and Enumerations

3.1 Setting up the Development Environment

3.1.1 Prerequisites

3.1.1.1 ACE/TAO

Miro is built on ACE/TAO. ACE/TAO must be installed in order to use Naming Service, Event Service, and Notification Service suppliers and consumers. Since Direct Connection suppliers and consumers depend on .NET Remoting and IIOP.NET they will still work without having ACE/TAO installed. However, many of the objects that are provided by ulmMiro and drdcMiro also depend upon ACE/TAO. Therefore, ACE/TAO must be installed as a prerequisite in all cases.

ACE/TAO can be obtained at [10]:

http://www.cs.wustl.edu/~schmidt/TAO.html

The current version of netMiro has been developed using ACE 5.5.2 and TAO 1.5.2.[9]

3.1.1.2 IIOP.NET

IIOP.NET is an essential prerequisite for netMiro in all cases.

IIOP.NET can be obtained at [12]:

http://iiop-net.sourceforge.net/

The current version of netMiro has been developed using IIOP.NET 1.9.0 beta1.

3.1.1.3 ulmMiro and drdcMiro

ulmMiro and drdcMiro are not prerequisites required by netMiro. Neither of these software packages needs to be installed or referenced for netMiro to compile and run successfully, since netMiro effectively duplicates the essential functionality.

However, as discussed below (Section 3.1.3), it is desirable to have at least the idl directories accessible to the netMiro installation, in order to generate interface libraries that netMiro can use to facilitate the creation of netMiro consumers for ulmMiro and drdcMiro suppliers, or to create Windows netMiro suppliers to replace existing Linux ulmMiro and drdcMiro suppliers.

3.1.2 Installing netMiro

Typically, netMiro will be provided as a .zip of the Visual Studio solution directory.

For the most seamless install, the unzipped netMiro should be located at: C:\code\base\netMIRO and an environment variable should be set to alias the installation directory; as follows:

```
set NETMIRO_HOME = c:/code/base/netMIRO
```

[9] At the time of this work (September 2006) a "work-around" patch was required to avoid a runtime error on the Notification Service when compiling with Visual Studio 2005. The workaround (Patch 11) was provided by OCI, for their ACE/TAO Version 1.4a and touched the files ProxyConsumer.cpp and ProxySupplier.cpp. The patch was expected to be integrated into the DOC Version of ACE/TAO "in the near future".

If the NETMIRO_HOME environment variable is not set correctly it will be necessary to modify the paths within the included batch files before they will run successfully.

Similarly, an environment variable called “IIOPNET_HOME” should be set to alias the installation directory for IIOP.Net.

Within the development environment (assumed to be Visual Studio .NET 2005) it may also be necessary to re-establish a correct reference to IIOPChannel.dll, if IIOP.NET is not installed at c:\code\base\IIOPNet.

3.1.3 Working with .IDL files to create Interface Libraries

IIOP.NET provides an extremely useful utility called “IDLToCLSCompiler.exe” that can be used to produce a Common Language Specification (CLS) .NET assembly from a collection of OMG Interface Definition Language (IDL) files. This assembly can then be referenced by other .NET programs.

netMiro uses this utility to produce a set of Interface Libraries from the collections of .idl files contained in the ulmMiro/idl, drdcMiro/idl, and netMiro/idl directories. These interface libraries provide the object references needed to produce consumers of those objects. Such .idl files provide the main means of exchanging interfaces between the Windows and Linux environments.

The netMiro Visual Studio solution calls a pre-build event that runs the batch file “PreBuildInterfaceLibraries.bat” whenever the netMiro project is compiled. This batch file makes the necessary calls to the IDLToCLSCompiler to produce the netMiroInterfaces.dll, ulmMiroInterfaces.dll, and drdcMiroInterfaces.dll and place them into the netMiro/InterfaceLibraries directory.[10]

3.1.4 netMiro Help

MSDN style help files for the netMiro namespace have been generated. These help files can be found in the netMiro/doc directory.

3.1.5 Examples

Examples have been provided for each of the following usage patterns:

Direct Examples (in netMiro\Examples\DirectExamples):

1. NetServer using a direct connection to NetClient and MiroClient over IPC (ExampleNETDirectIpc.Bat).

2. NetServer using a direct connection to NetClient and MiroClient over TCP (ExampleNETDirectTcp.Bat).

[10] Since this is usually unnecessary, unless an .idl file has actually been changed or added, it is often convenient to “rem” the pre-build event until it is really needed.

3. NetServer using a direct connection to NetClient and MiroClient over HTTP (ExampleNETDirectHttp.Bat).

4. NetServer using a direct connection to NetClient and MiroClient over IIOP (ExampleNETDirectIiop.Bat).

5. MiroServer using a direct connection to NetClient and MiroClient over TCP (ExampleMiroDirectTcp.bat).

6. MiroServer using a direct connection to NetClient and MiroClient over IPC (ExampleMiroDirectIpc.bat).

7. MiroServer using a direct connection to NetClient and MiroClient over HTTP (ExampleMiroDirectHttp.bat).

8. MiroServer using a direct connection to NetClient and MiroClient over IIOP (ExampleMiroDirectIiop.bat).

Naming Service Examples (in netMiro\Examples\NameServiceExamples):

1. NetNSServer using TAO Naming Service to communicate with NetNSClient and MiroNSClient (ExampleNetNS.bat).

2. MiroNSServer using TAO Naming Service to communicate with NetNSClient and MiroNSClient (ExampleMiroNS.bat).

Event Service Example (in netMiro\Examples\EventServiceExamples):

1. MiroEventPushSupplier using NetMiroEventServer to communicate with a MiroEventPushConsumer (ExamplePushEvent.bat).

Notification Service Example (in netMiro\Examples\NotificationServiceExamples):

1. NotifySupplier using NetMiroNotifyServer to communicate with a NotifyConsumer (ExampleNotifyEvent.bat).

Each of these examples is designed to be used as a starting template to produce operational suppliers and consumers (Table 1). The recommended practice is to copy the entire example directory of the appropriate type to a new project, and make modifications as necessary.

Table 1 – Usage Examples for use as Production Templates

(Console Apps)	Suppliers	Consumers	Services
Direct Connections	NETServer MiroServer	NETClient MiroClient	
Naming Service	NetNSServer MiroNSServer	NetNSClient MiroNSClient	NSLister
Event Service	EventPushSupplier	EventPushConsumer	NetMiroEventServer
Notification Service	NotifySupplier	NotifyConsumer	NetMiroNotifyServer

In addition, several practical Windows GUI based applications have been provided to demonstrate the use of netMiro classes:

1. netMiroLogger – an application to simultaneously monitor multiple Notification Service suppliers, while both recording and playing back log files of received events to multiple Notification Services. (Figure 8) Sources and Sinks are loaded using .xml files, and logs are in binary format.

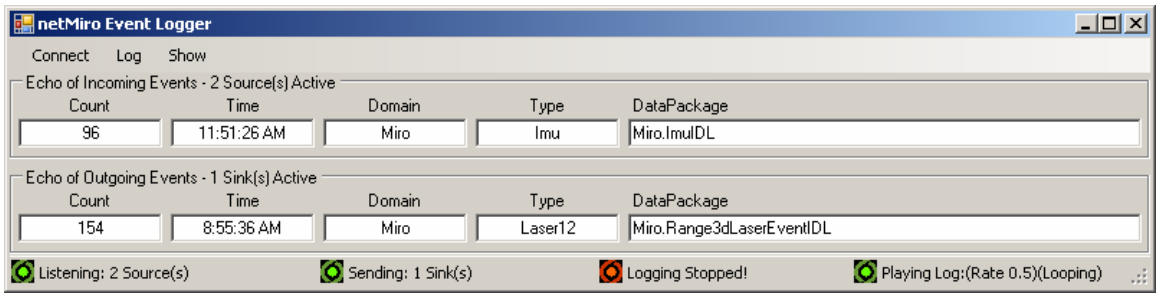

Figure 8 – Screenshot of the NetMiroEventLogger Application.

2. netMiroServiceMonitor – an application to monitor a list of Naming Services, Event Services, and Notification Services. It will indicate which of these services are active. If an active Naming Service is selected it will display the current set of bindings for that service. If an active Notification Service is selected it will display the list of events being offered by that Notification Service, as well as indicate which of those events have consumers subscribed to receive them. (Figure 9) The list of services to monitor is loaded from an .xml file at runtime.

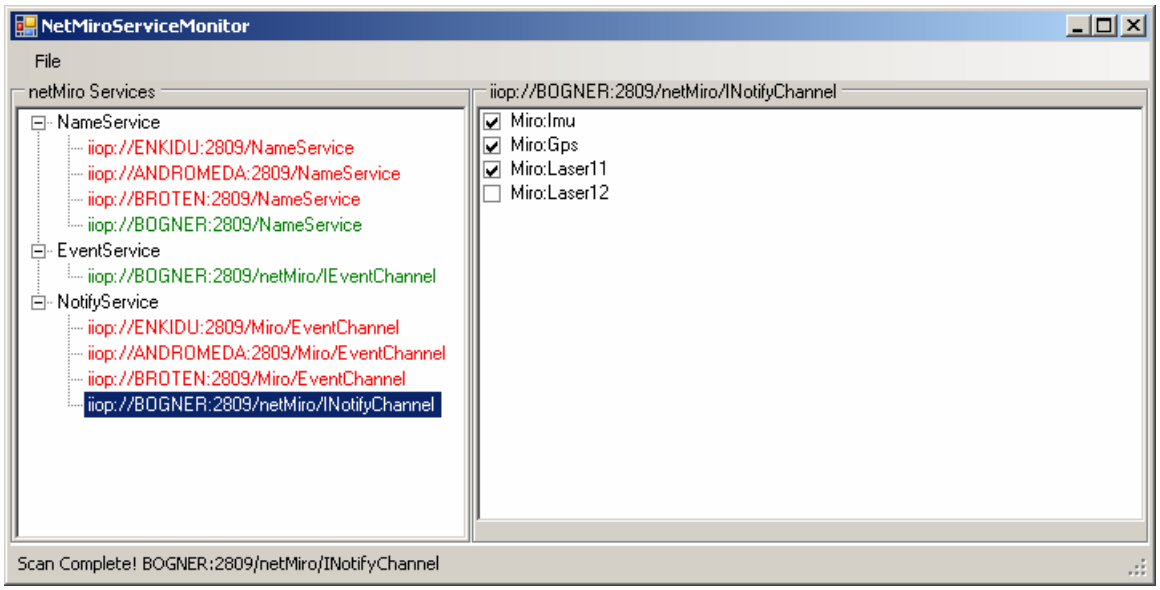

Figure 9 – Screenshot of the NetMiroServiceMonitor Application.

3.2 Using Direct Connection Suppliers and Consumers

3.2.1 Using Direct Connection Suppliers

3.2.1.1 NETServer

The NETServer console application demonstrates a very simple .NET Remoting server that uses .NET Remoting techniques to communicate with clients over TCP, HTTP, or IIOP channels. Only a single line (clearly marked in the example) needs to be changed to change the class of object being served:

```
Adder servantObject = new Adder();
```

By changing the references to "Adder" and "IAdder" to some other interface and implimentation, this code can be used for any other interface and implimentation defined in a referenced assembly. Alternatively, this code could be written to obtain the interface from a configuration file.

3.2.1.2 MiroServer

The MiroServer console application demonstrates a very simple netMiro server that constructs a netMiro.MiroDirectSupplier to establish communications and begin serving the object. As in the NETServer (Section 3.2.1.1 above) only a single line needs to be customized to change the class of the object being served:

```
Adder theServant = new Adder();
```

3.2.2 Using Direct Connection Consumers

3.2.2.1 NETClient

The NETClient console application demonstrates how to write a .NET client that obtains a local reference to a remote object using .NET Remoting techniques. It can obtain objects from both NETServer and MiroServer suppliers. The lines and methods that should be customized to consume different types of objects are clearly indicated in the example code.

```
IAdder servant = (IAdder)RemotingServices.Connect(typeof(IAdder), protocol + "://" +
host + ":" + port + "/" + objectURI);
```

By changing all references in this line of code from “IAdder” to some other interface the object/interface being consumed by this client can be changed. Methods that use the “servant” object can then be changed to ensure that the object is being used appropriately.

3.2.2.2 MiroClient

The MiroClient console application demonstrates the use of a MiroDirectConsumer to manage the communication between the client and the well-known server. The MiroClient can communicate with either a NETServer or a MiroServer. The regions of code that should be customized to consume different types of objects are clearly indicated in the example code.

```
InteractWithUser((IAdder)theMiroConsumer.Servant);
```

By casting theMiroConsumer.Servant to some type other than “IAdder”, a different object can be consumed. Obviously, the InteractWithUser() method would also have to be modified to ensure that the object was used appropriately.

3.3 Using NameService Suppliers and Consumers

3.3.1 Using NetNSServer

The NetNSServer console application demonstrates how to write a .NET server that registers its served object on a TAO Naming Service. A single line, which is clearly marked, needs to be customized to change the class of object being served:

```
Adder interfaceImplimentation = new Adder();
```

3.3.2 Using MiroNSServer

The MiroNSServer console application demonstrates how to write a netMiro server that uses the MiroNSSupplier class to register and bind its served object on the TAO Naming Service.

A single line, clearly indicated in the example, can be customized to serve different classes of objects:

```
Adder theServant = new Adder();
```

3.3.3 Using NetNSClient

The NetNSClient console application demonstrates how to write a .NET client that obtains a reference to a remote object from a Naming Service. The lines and methods that need to be customized to consume different classes of objects are clearly indicated in the example code.

```
IAdder servant = (IAdder)nameSpace.resolve(interfaceName);
```

The type of "servant" can be resolved to a desired type, and the InteractWithUser() method can be modified to accept the appropriate type of object as a parameter and to use that parameter appropriately.

3.3.4 Using MiroNSClient

The NetNSClient console application demonstrates how to write a netMiro client that uses the MiroNSConsumer class to obtain a reference to a remote object from a Naming Service. The lines and methods that need to be customized to consume different classes of objects are clearly indicated in the example code.

```
InteractWithUser((IAdder)theMiroUser.Servant);
```

Again, after casting theMiroUser.Servant to the desired type, the InteractWithUser() method should be modified accordingly.

3.3.5 Starting the TAO Naming Service

Miro (and netMiro) has been written to use the ACE/TAO framework. While netMiro has been written to use OMG standard CORBA, without reliance on ACE/TAO specifically, the examples provided assume that TAO is being used. However, it should work for any OMG Naming Service.

A batch file (StartNS.bat) has been provided to start the Naming Service. The command line for the batch file is the following:

```
StartNS.bat <string host> <int port>
```

To start the Naming Service directly, the command line is the following:

```
%TAO_ROOT%\orbsvcs\Naming_Service\Release\Naming_Service.exe -m 0
        -ORBEndpoint iiop://<host>:<port> -ORBDebugLevel 1
```

The –ORBDebugLevel switch controls the verbosity of the debug information that will stream to the console from the Naming Service (0-10; least to most).

3.3.6 Using the netMiro NSLister Utility

The netMiro NSLister utility is used to inspect the bindings that are currently active on a given Naming Service. (It performs the same type of function as the nslist utility provided by ACE/TAO).

A batch file (ShowNS.bat) is provided to start the NSLister utility. The command line for the batch file is the following:

```
ShowNS.bat <string host> <int port>
```

To start the netMiro NSLister utility directly, the command line is the following:

```
NSLister.exe <host> <port>
```

3.4 Using EventService Push Suppliers and Consumers

3.4.1 Using MiroEventPushSupplier

The EventPushSupplier console application demonstrates how to write a netMiro supplier that uses the MiroEventPushSupplier class to offer event driven data (which will typically be in the form of a structure defined in an .idl file) on an OMG Event Service. The lines of code and methods that need to be customized to change the kind of data being passed and the nature of the mechanism used to generate data-passing events on the supplier are clearly indicated within the example code:

```
private static netMiro.Examples.TestData testData;
…
testData = new TestData()
```

Change “TestData” to the desired class. In addition, the InteractWithUsersToRaiseEvents() method should be modified to perform appropriate interaction based on the desired class.

3.4.2 Using MiroEventPushConsumer

The EventPushConsumer console application demonstrates how to write a netMiro consumer that uses the MiroEventPushConsumer class to obtain event driven data from an OMG Event Service. The lines of code that need to be customized to change the kind of data being obtained and the nature of the event handling on the consumer are clearly indicated within the example code.

```
private static netMiro.Examples.TestData testData;
…
testData = new TestData();
```

In addition to changing the code above to refer to the desired class of testData; the pushConsumer_OnNewData() event handler should be modified to cast testData to the correct type, and perform appropriate interaction.

3.4.3 Starting the netMiroEventServer

netMiro provides its own event server (NetMiroEventServer), which uses the TAO Event Server implementation to bind an omg.org.CosEventChannelAdmin.EventChannel onto a running NameService at netMiro\IEventChannel.

A batch file is provided that starts the Event Service. The command line for the batch file is:

```
StartNetMiroEventService.bat  <string host> <int port>
```

To start the netMiro Event Service directly, use the following command line:

```
NetMiroEventService.exe –ORBInitRef
NameService=iiop://<host>:<port>/NameService -ORBDebugLevel 4
```

3.5 Using NotifyService Push Suppliers and Consumers

3.5.1 Using MiroNotifyPushSupplier

The NotifyPushSupplier console application demonstrates how to write a netMiro supplier that uses the MiroNotifyPushSupplier class to offer event driven data (in the form of StructuredEvent structures) on an OMG Notification Service. The lines of code and methods that need to be customized to change the kind of data being passed in the StructuredEvent.remainder_of_body property – which must be a structure of some kind (and which ought to be defined in a .idl file), and the nature of the mechanism used to generate data-passing events on the supplier are clearly indicated within the example code:

```
private static netMiro.Examples.TestData theDataPackage;
```

In addition to changing the class of “theDataPackage”, the InitializeStructuredEvent() method and the InteractWithUsersToRaiseEvents() method should be modified appropriately.

NotifyPushSupplier has been written to use an .xml initialization file to specify the specific types of events that the supplier will offer, as well as QoS properties (not implemented in the example).

3.5.2 Using MiroNotifyPushConsumer

The NotifyPushConsumer console application demonstrates how to write a netMiro consumer that uses the MiroNotifyPushConsumer class to obtain event driven data from an OMG Notification Service. The lines of code that need to be customized to change the kind of data being obtained and the nature of the event handling on the consumer are clearly indicated within the example code:

```
private static netMiro.Examples.TestData theDataPackage;
```

In addition to changing the class/structure of “theDataPackage” to the desired class/structure, the pushConsumer_OnNewData() event handler should be modified appropriately.

NotifyPushConsumer has been written to use an .xml initialization file to specify the specific types of events that the consumer wants to consume.

3.5.3 Starting the netMiroNotifyServer

netMiro provides its own notification server (NetMiroNotifyServer), which uses the TAO Notification Server implementation to bind an omg.org.CosNotifyChannelAdmin.EventChannel onto a running NameService at netMiro\INotifyChannel.

A batch file is provided that starts the Notification Service. The command line for the batch file is:

```
StartNetMiroNotifyServer.bat <string host> <int port>
```

To start the netMiro Notification Service directly, use the following command line:

```
NetMiroNotifyServer.exe -ORBInitRef
NameService=iiop://<host>:<port>/NameService -ORBDebugLevel
<debugLevel> -ORBRunThreads <numberOfServerThreads>
```

4 Conclusion

The netMiro package has been produced in order to facilitate the interoperability of Miro (including ulmMiro and drdcMiro) with suppliers and consumers running on the Windows platform. It supports .Net Remoting style Direct Connections, as well as CORBA Name Service, Event Service, and Notification Service suppliers and consumers.

Using netMiro it is possible to write Windows netMiro consumers for Linux Miro and Windows netMiro suppliers, and it is possible to write Windows netMiro suppliers for Linux Miro and Windows netMiro consumers.

5 References

[1] Naur, P., Randel B., (Editors); "Software Engineering: Report of a Conference Sponsored by the NATO Science Committee"; Garmish, Germany, 7-11 October 1968; Scientific Affairs Division, NATO; Brussels (1968).

[2] Krakowiak, S.; Quoted at "ObjectWeb – What's Middleware"; http://middleware.objectweb.org/ ; Accessed 29 July 2006.

[3] Broten, G., Monckton, S., Giesbrecht, J., Verret, S., Collier, J., & Digney, B.; "Towards Distributed Intelligence – A High Level Definition"; DRDC Suffield TR 2004-287; Defence R&D Canada – Suffield, (2004).

[4] Broten, G., Monckton, S., Giesbrecht, J., Collier, J.; "Software Systems for Robotics: An Applied Research Perspective"; International Journal of Advanced Robotic Systems; Vol. 3, No. 1 (2006); ISBN 1729-8806, pp. 11-16.

[5] Broten, G. and Monckton, S.; "Frameworks and Middleware for Unmanned Ground Vehicles"**;** In Proceedings of SPIE, Unmanned Ground Vehicle Technology VII, March 2005, Orlando, Florida, USA, Page 655-664.

[6] Enderle, S., Utz, H., Sablatnög, S., Simon, S., Kraetzschmar, G., and Palm, G.; "Miro: Middleware for Autonomous Mobile Robotics"; In Proceedings of IFAC Conference on Telematics Applications in Automation and Robotics; (2001).

[7] "The ADAPTIVE Communication Environment (ACE)"; http://www.cs.wustl.edu/~schmidt/ACE.html ; Accessed 03 October 2006.

[8] "Object Management Group"; http://www.omg.org/ ; Accessed 03 October 2006.

[9] "CORBA FAQ"; http://www.omg.org/gettingstarted/corbafaq.htm; Accessed 03 October 2006.

[10] "Real-time CORBA with TAO (The ACE ORB)"; http://www.cs.wustl.edu/~schmidt/TAO.html; Accessed 03 October 2006.

[11] "Miro – Middleware for Robots"; http://smart.informatik.uni-ulm.de/MIRO/index.html; Accessed 03 October 2006.

[12] "IIOP.NET – Overview"; http://iiop-net.sourceforge.net/; Accessed 03 October 2006.

[13] "Naming Service"; http://www.omg.org/technology/documents/formal/naming_service.htm; Accessed 03 October 2006.

[14] "Event Service"; http://www.omg.org/technology/documents/formal/event_service.htm; Accessed 03 October 2006.

[15] "Notification Service"; http://www.omg.org/technology/documents/formal/notification_service.htm; Accessed 03 October 2006.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for who the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in Section 8.)

Defence R&D Canada – Suffield
PO Box 4000, Station Main
Medicine Hat, AB T1A 8K6

2. SECURITY CLASSIFICATION
(overall security classification of the document, including special warning terms if applicable)

Unclassified

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title).

Introduction to netMiro: Extending Middleware for Robots (Miro) to the .NET Platform

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Bogner, Stephen

5. DATE OF PUBLICATION (month and year of publication of document)

December 2006

6a. NO. OF PAGES (total containing information, include Annexes, Appendices, etc) 31

6b. NO. OF REFS (total cited in document) 15

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Final Technical Report of work conducted from June 2006 to September 2006

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

Autonomous Intelligent Systems Section

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC Suffield TM 2006-187

10b. OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(x) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally corresponded to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected).

Unlimited

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C) or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Middleware for Robots (Miro) is an Open Source, Object Oriented "robotics-centric" software framework, which exploits the Adaptive Communications Environment (ACE) and the Common Object Request Broker Architecture (CORBA) built on it – The Ace Orb (TAO) - to provide the communications services necessary for distributed computation – a pervasive requirement in robotics.

After conducting a detailed review of available technologies for robotics middleware, Defence R&D Canada (DRDC) Suffield Autonomous Intelligent Systems Section (AISS) decided to exploit Miro within the DRDC program.

A design objective of any middleware technology is to support interoperability across heterogeneous platforms. Miro achieves this through the use of CORBA. Unfortunately, the fact that Miro has been implemented on Linux platforms, using ANSI C++ and Linux toolsets, for a target audience of Linux users, has created a barrier to its acceptance and use by programmers who are more comfortable developing software in the managed C# language using the .NET framework within the Windows environment.

The netMiro project was undertaken to extend the Miro technology to the Windows .NET platform.

This report provides a brief introduction to the netMiro software package.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifies, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Autonomous Intelligent Systems, Military Robotics, Middleware, ACE, TAO, CORBA, MIRO