

Towards a mobile agent framework for Nomad using .NET

JITEN RAMA
University of Pretoria
and
JUDITH BISHOP
University of Pretoria

Mobile agent systems offer a wide variety of advantages. However, most actual mobile agent systems only offer a wired connection and typically have very limited, if at all, mobile device connectivity features. To date, Java has been the foundation of almost all existing mobile agent systems. This paper discusses the approach to a “pluggable” mobile agent system for Nomad, developed with Microsoft .NET and Microsoft .NET Compact framework. Nomad is intended for use on a variety of .NET wired and wireless devices (e.g. Pocket PC) that support Bluetooth, Infra-red and 802.11, which are fast becoming the norm for ad hoc networks. The intentions of the Nomad system and the intended use of mobile agents in the system are discussed. It is shown that mobile agents are perfect for nomadic environments and would be of a great advantage to the Nomad system.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design - *Wireless communication*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems - *Distributed applications*; C.5.3 [**Computer System Implementation**]: Microcomputers - *Portable devices*; D.1.3 [**Programming Techniques**]: Concurrent Programming - *Distributed programming*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - *Search process; Selection process; Retrieval models*;

General Terms: Languages, Algorithms, Design, Verification

Additional Key Words and Phrases: Mobile agent system, Nomad, .NET, .NET Compact Framework, Wireless connectivity

1. INTRODUCTION

The concept of mobile agents has been around for a few years. Mobile agents have been used in industry for many reasons and many situations. So what is a mobile agent exactly? According to Danny B. Lange [1998]:

‘A mobile agent is not bound to the system where it begins execution. It has the unique ability to transport itself from one system in the network to another. The ability to travel allows a mobile agent to move to a system that contains an object with which the agent wants to interact and then to take advantage of being in the same host or network as the object.’

Mobile agents need a mobile agent system to reside in. The mobile agent system hosts one or more mobile agents and has the ability to send and receive mobile agents from another mobile agent system. Furthermore, the mobile agent system allows access to resources on the system which the mobile agent may use.

Almost all mobile agent systems have been designed and written using Java. A few examples of these are Aglets [Lange, 1998], NOMADS [Suri, 2000], JADE and μ Code¹. There have been some initiatives to produce mobile agent systems using the Microsoft .NET Framework, namely MAPNET [Staneva, 2004] and EtherYatri.NET. Some systems, such as CARLA [Aleksy, 2003], have also explored the use of wireless mobile agents in handheld devices.

In this paper, Nomad is described as an alternative design that encapsulates both the Microsoft .NET Framework, and the Microsoft .NET Compact Framework. Although Nomad’s mobile agent framework is more of a “customized” version of current systems, there are fundamental similarities and differences in what the system offers.

The paper serves as an introduction to Nomad and how the intended Nomad agent framework would “plug” into the Nomad system. In section 2, the Nomad system is discussed. Section 3 discusses the relevant points of the .NET Framework and Compact Framework. Section 4 describes what the agent system intends to achieve. In section 5, an overview of how the systems would “plug” together is discussed and a comparison to other systems is made. Section 6 concludes this paper.

¹ Systems with a primary reference as a website are listed under the project name in the website references section at the end of the paper.

Author Addresses:

Jiten. Rama, University of Pretoria, South Africa; genx@telkomsa.net.

Judith. Bishop, Department of Computer Science, University of Pretoria; jbishop@cs.up.ac.za.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, that the copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than SAICSIT or the ACM must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 2004 SAICSIT

2. NOMAD

Nomad is intended to be used by small communities that wish to share information. Nomad works on the basis of artefacts in a distributed environment. The users themselves are casually connected to a network and hence the community. The users could be in different time zones, have more than one device on which the information could reside and use a variety of protocols, wired or wireless, to connect to the community. They also tend to move around, for example, from home to office and on and off airplanes. In other words, Nomad users are of no fixed address.

Nomad provides the capability for a number of projects to reside on a single system. Figure 1 shows a Nomad project directory that resides on a computer that belongs to the user, Felix. Felix is part of two projects, namely project X and project Y. In project X, there are three users, namely, Juan, Felix and Nina. Juan enjoys travelling, so he has three IP addresses, the last of which was a wireless connection in the Johannesburg airport in South Africa. Nina on the other hand, has two workstations, one based at home, and the other is a laptop at work. Simone, who is part of project Y, has 2 devices which use Nomad. The first device is her workstation. The other device is a Pocket PC which runs a “lightweight” version of Nomad (discussed in section 3.2). Simone occasionally connects to the community in the coffee shop that offers wireless internet to see if any updates have been made to the artefacts she needs to continue with her work. If no updates are available, she has a while longer to enjoy her coffee.

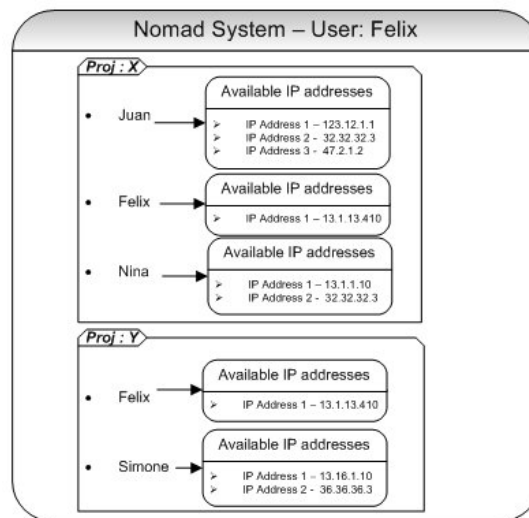


Figure. 1. Nomad Project Structure.

Nomad has the capability to “clean” a user’s IP address list, if the address has not been used in a while. Each user has to define a default “permanent” IP address at project inception, although Nomad has the ability to regain the user IP address in the situation that the user has continuous changing IP addresses, with appropriate user interaction.

Nomad should not be confused with a version control system. If a user seeks certain artefacts, then Nomad will search the community, and according to the user’s preferences, give a list of available artefacts or get the best possible match of the artefact. Artefacts are delivered to the user, who then makes an informed decision, and adds it to the project that they are working on.

An example would be a book that is written by multiple authors. Each author would have the Nomad system running on their computers, and if they needed certain updated artefacts, they would activate Nomad, which would take the ‘shopping-list’ that contains the needed artefacts and look for the updates on each Nomad system running the same project. If a certain user is not available, i.e. the user cannot be reached on any of the available IP addresses in the list, Nomad would make a note and if the user prefers, as soon as the user becomes available, Nomad would gather any information that was required from the shopping list.

3. .NET AND THE COMPACT FRAMEWORK

Although Java has till now been the mainstream developers’ choice for distributed applications, the introduction of the .NET framework is bound to change that. Within a few years of its release, .NET, along with the C# language, has

altered the mindset of programmers. This section mentions a few of the many features that have a direct impact on how designers develop distributed systems. Furthermore, with the release of the .NET Compact Framework, developers have access to APIs which allow programs to be developed on mobile devices, provided that they support the framework. The framework has advantages and disadvantages, some which will be discussed in this section.

3.1 .NET framework

The .NET Framework is a platform that simplifies application development in the highly distributed environment of the Internet [MSDN]. As opposed to the virtual machine in Java, the .NET version of a *virtual machine* is the CLR. The topic of the CLR (Common Language Runtime) is not within the scope of this paper, though it is necessary to mention it, as it serves as the backbone to .NET. Some of the features as noted by [Robert, 2001] follow:

- The CLR simplifies development. The CLR is responsible for the “nitty-gritty” implementation of code. Memory management is handled by the garbage collector. Metadata allows dynamic binding of executables. In addition, reliability is enhanced through type safety. All details of structure size and the organization of members within an object are kept, so there is never a need to worry about alignment or packing issues. Furthermore, boundary checking for arrays or buffers is automatic.
- The CLR works hand in hand with tools (such as Visual Studio), with compilers, debuggers and profilers to make the developer’s job much simpler.
- The CLR makes multiple language support a reality. The basis for multiple language support is the Common Type System and metadata. The basic data types used by the CLR are common to all languages. There are therefore no conversion issues with the basic integer, floating point, and string types. All languages deal with all data types in the same way. There is also a mechanism for defining and managing new types. All top-level languages compile to IL(Intermediate Language). Once the compilation is made and the metadata is generated for the object, the code is easily accessible from other languages. Therefore it is possible to write a class in VB and inherit from it in C#.

3.2 Remoting - System.Runtime.Remoting

Remoting is a framework that is built on the CLR for building distributed applications in an object oriented way. Client applications can invoke functions and access resources on a server object, be they on the same computer, or a remote computer over a network, or another application domain in the same process. Communication between the client and server object is channeled through a proxy object. The proxy allows the client to make calls to the server, using function calls that seem to be local to the client.

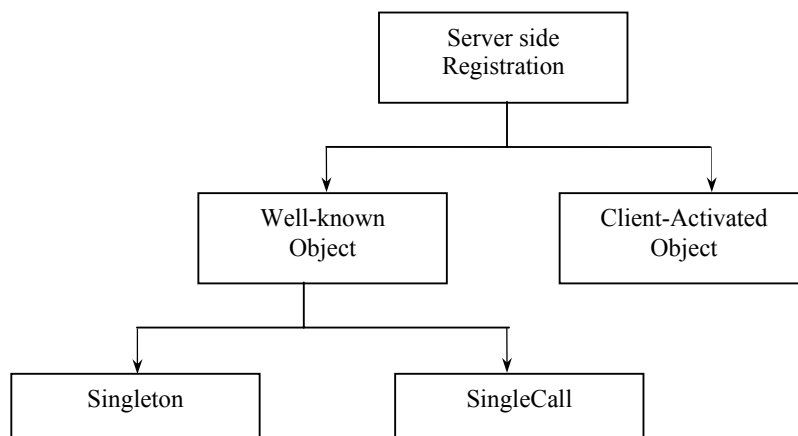


Figure. 2. Remoting Server Object Types.

There are two types of server object types that .NET supports: *well-known objects* and *client-activated objects*, as shown in Figure 2. Communication with a well-known object is established each time a message is sent by the client, whereas with a client activated object, there is a permanent connection until the client is satisfied.

Well-known objects are server-activated objects. Well-known objects are either singleton or singlecall. With a singleton, *all* messages from *all* clients speak to the same single object running on the server side. On the other hand,

with a singlecall, *each* message from any client is processed by a new object running on the server. If the well-known object was a bank, then a singleton would relate to *all* clients talking to the same personal banker each time they visit the bank, whom would handle *all* queries. A single call would then relate to *each* client having to talk to someone new *each* time they had to query something at the bank. It should be noted that well-known objects must have *parameterless* constructors.

Client-activated objects are normally used by programmers creating dedicated servers, which provide services to the client, which they are also writing. The client and server create a connection, based on a lease time, and then maintain the connection until the needs of the client are satisfied or the lease time expires.

The *System.Runtime.Remoting* namespace provides the developer with the needed classes for remoting objects. Remoting would form the mode of transport for mobile agents between mobile agent platforms. Using remoting reduces the need for low level socket programming. The concern with remoting is that firewalls could pose a problem, although the use of a HTTP channel over a TCP channel could simplify the solution. HTTP has been noted for its ability to get through firewalls.

3.2.1 Reflection - *System.Reflection*

The need for a mobile agent to know its identity is crucial when moving between the agent platforms. Reflection allows a program to collect and manipulate its own metadata. *Metadata* is the key to a simpler programming model, eliminating the need for Interface Definition Language (IDL) files, header files, or any external method of component reference. Metadata allows .NET languages to describe themselves automatically in a language-neutral manner, unseen by both the developer and the user [MSDN]. From the assembly, the identity can be discovered.

The agent would become available dynamically when it arrives at the next platform. Reflection would allow the application to instantiate an agent object when it becomes available dynamically. Furthermore, the methods for starting the agent can be handled by dynamic function invocation. The method *Registerwellknownservicetype* uses reflection to build a proxy for an object on the server side. Furthermore, *System.Reflect.Emit* supports dynamic creation of new types at runtime.

3.2.2 Other technologies

Events and delegates are closely associated in the .NET framework.

‘An event is a message sent by an object announcing that something important has happened. Events are implemented using delegates, a form of object-oriented function pointer that allows a function to be invoked indirectly by way of a reference to the function.’ [MSDN]

Events and delegates would form an integral part of the implementation of Nomad. An agent would react to events and respond using a delegate to execute its reaction.

An agent would need to be serialized to be able to move between systems.

‘Serialization is the process of taking objects and converting their state information into a form that can be stored or transported. The basic idea of serialization is that an object writes its current state, usually indicated by the value of its member variables, to persistent storage. Later, the object can be re-created by reading, or deserializing, the object's state from the storage. Serialization handles all the details of object pointers and circular object references that are used when you serialize an object.’ [MSDN]

The .NET framework offers three predefined types of formatters: XML, binary and ActiveX. The choice of formatter used for an agent would be that of binary, although for use on devices, agents would use XML serialization since the compact framework does not offer binary serialization.

3.3 .Net Compact Framework

Devices such as the Pocket PC, have become main stream in the recent years. Due to their small “pocket” size, they can be taken just about anywhere making them convenient in this day and age where people are constantly on the move. They come with a variety of “lightweight” versions of desktop applications built in, e.g. Microsoft Office, email clients and Adobe Acrobat Reader. They come with a variety of connectivity standards, both wired e.g. USB (universal serial bus), or wireless, e.g. Bluetooth, 802.11(wireless LAN or WiFi) and Infra-red connectivity options.

It would therefore make perfect sense to have these devices included into Nomad. Although with time, comes greater storage capacity and increased processor speeds, the devices currently are limited in memory, space and performance. The .NET Compact Framework is a version of .NET specifically designed for devices with limited memory, space and performance. As Makofsky [2004] states:

‘The class library provided by the Compact Framework is extremely similar to its desktop counterpart, except that certain functionality has been “slimmed down” (or entirely eliminated) to better support the limited memory, storage space and performance of a mobile device.’

The following sections mention a few class libraries that would be of importance to the Nomad project.

3.3.1 Lightweight Nomad

It would have been the perfect scenario to be able to run the Nomad desktop version on devices such as a Pocket PC. Unfortunately, a lightweight version is needed due to the following:

- Although the compact framework allows WiFi connections by simply making IP connections, there is currently no Bluetooth support in either the compact or the full framework.
- Remoting and binary serialization, both of which are of major significance to the desktop version of Nomad, are currently not supported on the mobile platform.

With the release of this platform’s SDK, some Bluetooth features have been included with the windows sockets, but only for the desktop version.

Functionality for Bluetooth, remoting and serialization will have to be handled in a different manner for versions running Nomad on mobile devices. A lightweight version of Nomad will therefore include only some of the functionality of the desktop version.

CARLA [Aleksy, 2003] has produced a similar lightweight concept in CORBA. The CARLA (*CORBA-based Architecture for Lightweight Agents*) project focused on the minimum CORBA specification, which could be related to the Compact Framework offered by .NET, whereby certain features were minimized to support devices with limited memory, space and performance.

3.3.2 Networking functionality

The compact framework provides a *System.Net* namespace which contains all the needed classes to create networked applications.

Name	Description
Dns	This class provides simple domain name resolution functionality.
IPHostEntry	This class handles internet host address information.
EndPoint	An abstract base class used to represent a network resource or service.
IPAddress	This class provides an Internet Protocol (IP) address
IPEndPoint	This class represents an EndPoint with port information
IrDAEndPoint	This class represents an EndPoint that represents a network infrared service.
SocketAddress	This class stores serialized information from EndPoint derived classes
Sockets	This namespace provides a managed implementation of the winsock interface.
WebRequest	An abstract base class used for making a request to a URI(Uniform Resource Identifier)
WebResponse	An abstract base class used to provide a response to a URI(Uniform Resource Identifier)

Table 1. Objects of interest in the *System.Net* namespace.

The scope of this paper does not allow for in depth discussions of each of these classes. Table 1 shows a few important objects that will aid in the production of Nomad. These classes would provide the lightweight Nomad with the ability to communicate in most ad hoc networks.

Figure 3 shows two code listings. Code listing (a) would be used in a case where the device only is aware of the hosts’ domain name, in this case ‘*www.cs.up.ac.za*’. The *System.Net.Dns* namespace can resolve the domain name. Once the IP address has been obtained, we create an endpoint, which is a combination of the IP address and the port number, in this example, port 80.

Code listing (b) demonstrates how a device can identify its own IP address. This feature can be used at runtime when the device has continuously changing IP addresses.

<pre> /* Resolve the UP CS Web Server IP address */ System.Net.IPEndPoint cs_server = System.Net.Dns.GetHostByName("www.cs.up.ac.za"); /* Create the endpoint */ System.Net.IPEndPoint cs_Endpoint = new System.Net.IPEndPoint(cs_server.AddressList[0], 80); string cs_ip_add = cs_Endpoint.Address.ToString(); </pre>
<i>(a) Resolving an IP address</i>
<pre> string hostname = System.Net.Dns.GetHostName(); System.Net.IPEndPoint ipAdd = System.Net.Dns.Resolve(hostname); string hostname_ip_add = ipAdd.AddressList[0].ToString(); </pre>
<i>(b) Identifying an IP address</i>

Figure 3. Usage of the System.Net namespace.

4. A MOBILE AGENT SYSTEM

4.1 Why agents?

Nomad would be a perfect system to make use of a mobile agent system. Mobile agents work well in casually connected environments. Danny B. Lange, [1998] states that there are seven good reasons to use mobile agents.

1. They reduce network load.
2. They overcome network latency.
3. They encapsulate protocols.
4. They execute asynchronously and autonomously.
5. They adapt dynamically.
6. They are naturally heterogeneous.
7. They are robust and fault tolerant.

Although not all the above are applicable to Nomad, the few that make a difference are discussed.

Firstly, agents reduce network load and have the ability to execute asynchronously. There will be occasions when substantial amounts of data will be moved across the network, or among the community. In this case, the user might want to send an agent to find available artefacts, and then view the results at a later time. It therefore makes sense to move the processing to the data, not the data to the processing. In Figure 1, Juan might decide to look for updated artefacts at the airport, but might not have enough time to get the results since he has to board the plane. By instantiating the project “compile”, he has sent out the mobile agents to all the available nodes. The nodes continue in an autonomous manner to collect the needed data. When Juan becomes available to the community once again, the agents would be sent home, to Juan’s system with the results.

The fact that they are heterogeneous, robust and fault tolerant, allows the agents to make decisions based on their environments on which they are executing. If the computer is about to shut down, the agent will be aware that it should save its state so as to persist on startup. It might even make the choice to move to the node of origin based on user preferences. Since Nomad is based in the .NET framework, it will have the ability to run on any operating system that supports the .NET environment. This applies to devices that run on the .NET compact framework as well.

The use of .NET Remoting (section 3.1.1) allows the agent to be independent of underlying protocols. Hence, in a way, Nomad agents encapsulate the needed protocol used to move between systems.

4.2 Ad hoc networks

With the increase of devices, ad hoc networks are becoming more and more common. An ad hoc network is simply a local network connection between two or more nodes, where the connection itself is created dynamically. They are

decentralized, and each node in the network is responsible for all aspects of their network connection, including establishing connections and maintaining routing protocols. The physical location of the node determines the topology of the network [Quirolgico 2002].

Simple examples of these networks are Bluetooth, 802.11 and Infra-red. Devices that use these network connections have the ability to “look” for each other and set up the network connection without much interaction from the user. Due to the fact that these nodes are mobile and move in and out of network range, Nomad should take care to handle loss of connectivity well.

When Nomad is used in an ad hoc environment, care will be taken to handle fragile connections. The use of agents in ad hoc networks makes sense. Nomad would rely on the fragile connections merely for sending the agent across and later to be retrieved, thereby allowing the mobile agent to work autonomously on behalf of the user.

5. NOMAD AND THE MOBILE AGENT SYSTEM.

5.1 Towards a pluggable system

The intention of Nomad is not to be an agent framework, or an agent toolkit. Nomad will make use of mobile agents in ways that would be better than using a classical client server architecture. Nomad’s main aim is to provide a small, closed, community of users with nomadic tendencies, the ability to share information in a way that would make life simpler. We aim to use agents as a pluggable system, that is, use a standard interface to the main Nomad system. This would allow comparisons at a later stage on whether agents actually do work better than classical client-server architectures.

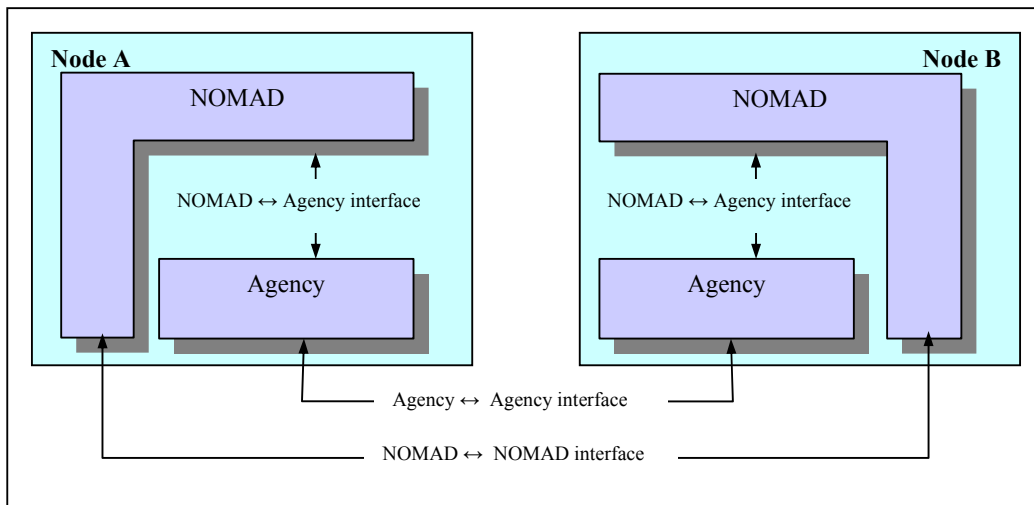


Figure 4. Mobile agents as a pluggable system to Nomad.

Figure 4 shows the intention of a “pluggable” system. The Nomad system would be able to talk via the agencies or between Nomad systems themselves.

5.2 Comparison to other similar systems

An important aspect of our work has been the investigation of other mobile agent frameworks for nomadic communities. Table 2 illustrates the similarities between other systems when compared to Nomad. Each system has specific intentions to why they were created.

Reference	Name	Description	Language / Architecture	Wireless Connectivity	Other
Lange, 1998	Aglets	API from IBM used to program mobile internet agents	Java	No	Free, Open source, Educational
Aleksy, 2003	CARLA	CORBA based Lightweight agents.	Java and CORBA	Yes	Aimed at providing an architecture for handheld devices.
Delamaro, 2002	μ Code	Mobile code toolkit	Original in Java with .NET ported version available.	No	Open source, Educational.
Quiroigico, 2002	SWAN	SWAN provides a test-bed for devices that make use of ad hoc networks.	Java	Yes	Uses JADE[JADE] as underlying Agent framework
Staneva, 2004	MAPNET	Mobile Agent Framework	C# and .NET	No	Follows the MASIF specification. Educational.
Suri, 2000	NOMADS	Mobile agent system that supports strong mobility.	Java	No	Composed of 2 parts, execution environment (OASIS) and Aroma – compatible Java VM that captures state info.
MONADS and HEIKKI 1999	Monads	Research project aimed to support nomadic users in the near future.	Java	Yes	Most closely related to Nomad.
ETHER YATRI .NET	EtherYatri.NET	Mobile agent toolkit	C# and .NET	No	Free, Open source, Educational. Integrates into Visual Studio
NOMAD	Nomad	Aims to support nomadic users. Use of mobile agents to add certain functionality.	C#, .NET and the .NET Compact Framework	Yes	Not a Mobile agent toolkit, nor a framework, but rather makes use of mobile agents.

Table 2. Nomad in comparison with other projects.

Aglets formed the basis of major mobile agents. It is the oldest system in this comparison. The reason for it being included in this comparison is that although mobile agent systems have improved in many ways, Aglets were always mentioned as a comparable system. The architecture and ideas that Aglets brought to the mobile agent community still form the basis of current systems.

The intention of CARLA is to provide a lightweight version of the bigger framework CORBA. The minimum CORBA specification is investigated for the reason that current devices would not be able to run the full CORBA framework. Similarly, Nomad intends to make use of a lightweight version of the .NET, the Compact Framework to support such devices.

SWAN (*Simulator for Wireless Ad-hoc Networks*) makes use of the JADE (*Java Agent Development Framework*) mobile agent framework. This is a perfect example of agents at work. SWAN along with the WAS (*Wireless Agent Simulator*) were written in Java and was implemented on top of the FIPA-based JADE [Quiroigico 2002]. It should be noted that these are wireless agents and not wireless *mobile* agents, where the latter migrates between hosts. Agents resided above the WAS and made use of resources provided by the virtual device. SWAN provides functionality to other wireless agents and itself, but does not enable them to be migrated across host boundaries, as is the intention of Nomad.

For mobile agents to move between hosts, agents have to capture their state. By allowing the state to persist, the agent is able to continue execution when it is revived once again on the receiving host. NOMADS has the ability to capture and transfer the full execution state of mobile agents, hence this system supports *strong mobility*. Furthermore, NOMADS provides safe agent execution. These features are provided by an agent execution environment, called *Oasis* and a new Java compatible virtual machine called *Aroma*. Nomad has no intentions to recreate a virtual machine for its agents, although the execution of agents will be held within the confinements of the Nomad agent system.

Although there are mobile frameworks available in .NET, none has functionality of wireless connectivity. EtherYatri.NET and MAPNET are two such systems. Both these systems allow facilities for user-definable agents. Nomad is a system that makes use mobile agents. Users will not be able to create agents themselves. Agents might encapsulate user preferences, but in no way are user-defined.

Monads examine adaptation agents for nomadic users. The Monads architecture is based on the Mowgli communications architecture that takes care of data transmission issues in wireless environments. Therefore Monads extends existing systems with mobility-oriented features and is not a new agent system. Monads provide support for mobile devices, but extend the use of devices to mobile cell phones. Nomad will support devices that are used by nomadic users that run on .NET compact framework, but Nomad has no intention at this point to provide support for cell phone users.

As can be seen, Nomad is comparable to most systems, and intends to achieve the combination of some of the systems. Support for nomadic users is what Nomad intends to achieve.

6. CONCLUSIONS

In this paper we presented Nomad. The Nomad system is aimed to be used by small communities who would like to share artefacts, be it pictures, text or documents. They would connect to the community by any available means, wired or wireless, and would like to do this hassle free. The lightweight version of Nomad would allow devices with small memory and processor footprints, to seamlessly connect to other users with the same ease of use.

Most mobile agent systems are Java based. Nomad intends to offer an alternative approach by using .NET and the .NET compact framework. The .NET framework offers many features which Nomad would rely greatly on. This paper has highlighted a few of these features, namely remoting, reflection, events and serialization.

This paper further discusses the intended use of a “pluggable” mobile agent framework that would be responsible for some of the main features in Nomad. Mobile agents have been shown to provide ease of use, with minimum user intervention. We discuss the similarities and difference of other available systems and note that Nomad offers a combination of related features.

We believe that the marriage of Nomad and Nomad’s mobile agent framework would allow nomadic users to be more concerned with important matters, rather than file sharing.

7. ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support of Microsoft Research and the South African Department of Trade and Industry’s THRIIP Programme, Grant No: 2788. Members of the Nomad team, especially Ronald Klazar and Tebalo Tsoaeli have helped with many of these ideas.

8. REFERENCES

- ALEKSY, M., KORTHAUS, A., SCHADER, M. 2003. Design considerations for a CORBA-based architecture for lightweight agents (CARLA). *Web Intelligence and Agent Systems: An international journal*, 259-271
- DELAMARO, M & PICCO, G.P. *Mobile Code in .Net: A Porting Experience.*, Mobile Agents, 6th International Conference, (pp. 16-31). MA 2002: Springer
- HEIKKI, H., HEIMO L., & KIMMO R. 1999. *Mobile Agent Communication in Wireless Networks*. In Proceedings of the European Wireless'99/ITG'99. pp. 211-216, October 1999
- LANGE, D. B., AND OSHIMA, M. 1998. *Programming and Deploying Java Mobile Agents with Aglets*. Reading MA: Addison- Wesley
- MAKOFISKY, S. 2004. *Pocket PC Network Programming*, Addison Wesley Professional.
- QUIROLGICO, S., et al. 2002. Wireless Agents in Ad Hoc Networks, *Innovative Concepts for Agent-Based Systems, First International Workshop on Radical Agent, WRAC 2002*, W. TRUSZKOWSKI, C. ROUFF AND M. HINCHEY, Eds. Springer, Verlag Berlin Heidelberg, 165-174.
- ROBERT, P., AND WEEKS, R. 2001. *C# and the .NET Framework, The C++ Perspective*. SAMS, Indiana, USA.
- STANEVA, D., AND DOBREVA, D. 2004. MAPNET: A .Net-Based Mobile-Agent Platform. *International Conference on Computer Systems and Technologies - CompSysTech' 2004*
- SURI, N., BRADSHAW, J.M. AND BREEDY, 2000 M.R., *An Overview of the NOMADS Mobile Agent System*, In Proceedings of ECOOP'2000, Nice, France, 2000.

8.1 Website References

- μCODE <http://mucode.sourceforge.net>
.NET Compact Framework discussion http://msdn.microsoft.com/chats/transcripts/mobileembedded/embedded_101404.aspx

802.11	http://standards.ieee.org/wireless/index.html
BLUETOOTH	http://www.bluetooth.org/
ETHER YATRI .NET	http://www.geocities.com/siddharthuppal/EtherYatri.htm
IR	http://www.irda.org
JADE	http://jade.tilab.com/
MONADS	http://www.cs.helsinki.fi/research/monads/
MSDN	http://msdn.microsoft.com
NOMAD	http://polelo.cs.up.ac.za/nomad/