The design of a protocol for collaboration in a distributed repository - Nomad

by

Jiten Rama

September 2006

Submitted in partial fulfillment of the requirements for the degree Magister Scientiae (Computer Science) in the Faculty of Engineering, Built Environment and Information Technology University of Pretoria

The design of a protocol for collaboration in a distributed repository - Nomad

by
Jiten Rama

Computer Supported Cooperative Work (CSCW) is the study of how people use technology, with relation to hardware and software, to work together in shared time and space. Mobile office environments are becoming commonplace. Workers form virtual online communities on a global scale and use groupware to collaborate and complete a common goal. We tend to be mobile, yet need to be available to collaborate.

This thesis investigates a protocol for our decentralized artifact control system, Nomad. Nomad enables globally dispersed members of small casually connected communities to share artifacts which are gathered on a best effort approach. The Nomad protocol takes into consideration the work habits of users and their variety of devices.

The major contribution of this thesis is a simulator of the Nomad protocol, which serves as a proof-of-concept for its design. Specifically, we look at how such a protocol handles casually connected small communities. We consider high level aspects such as setting up the community, the overhead of nodes, availability, scalability and connectivity. We demonstrate scenarios that the protocol will need to handle. Furthermore, we take a broad look at CSCW, push and pull technologies, peer-to-peer technologies, and enabling technologies such as Microsoft .Net. These form the basis of the Nomad design. In addition, we suggest the integration of mobile agents, which we consider a future addition to Nomad.

It was found that the protocol had to compensate for two nodes that were never online at the same time. In the case that a best effort approach is not feasible, we propose alternate approaches at the cost of overhead on a propagation node. The developed concept provided valuable insight into the problem domain, outlined the boundaries of the protocol and provided a possible solution for Nomad. The simulator proved to be a useful tool for determining outcomes from possible scenarios. The results from the simulator will feed directly into the development of Nomad.

Keywords: CSCW, groupware, information sharing, distributed applications, simulation, push and pull, communities, P2P, mobile agents, Microsoft .Net.

Supervisor: Prof. J. Bishop
Department: Department of Computer Science
Degree: Magister Scientiae

Acknowledgments

Dedication hardly ever comes without inspiration. Inspiration is a force which closes the gap between those that prosper and those that do not. There have been many that have inspired me along this journey which has now come to an end. Friends and family, too many to mention, to all those who have inspired me when hope had failed me, I thank you. I'm sure that each of you will undoubtedly know who you are. That being said, there are those that should take special mention.

Special thanks go to my supervisor, Professor Judith Bishop for her continuous support and leadership with her insightful advice from the start to the end of this journey. Thank you for boosting my confidence when it needed to be boosted. Thank you for significantly improving this thesis in terms of focus, readability and details. I cannot fully express my gratitude for your generosity, confidence and superb guidance.

To my parents, Manilal and Urmilla, who have always been by my side. To my father who always lead me one step forward knowing that I had learnt life lessons from the step before. To my mother who never quite knew what this thesis was about, offered emotional support at any given time. To my parents, to whom I owe my very existence and wellbeing, I dedicate this.

To Himal, Janita, Divyesh & little Meeshaal whose encouragement paved a path on which I walked. I gratefully acknowledge the contributions of other Nomad members.

Beryl Stiles whose task it was to grammatically correct the flow of words that makes up this thesis. I am deeply grateful for your kindness that only angels possess. Thank you for time and effort in proofreading this thesis.

To Vishani, I thank you for your patience and understanding. We spent much time apart during this period of time. Your friendship and love have withstood the test of a thesis, and to that I am immensely grateful.

Table of Contents

List of Figures

List of Tables

Papers from this thesis

Jiten Rama and Judith Bishop, *A Survey and Comparison of CSCW Groupware Applications,* Proceedings of South African Institute of Computer Scientists and Information Technologists (SAICSIT) 2006, 198 - 205, 2006.

Jiten Rama and Judith Bishop, *Simulating a protocol for a casually connected distributed artifact repository*, submitted October 2006 to IEEE Transactions on Mobile Computing.

1. Introduction

1.1. Background to Nomad

In this section we give a brief overview of Nomad. We highlight the project's history, intentions and objectives.

Nomad is a framework for a distributed resource management system, with special emphasis placed on the accessibility of information stored on detached devices, such as personal computers, laptops, PDA's and flash-disks. Nomad is intended to address the emerging problem of information spread and neglect, where users have the same copy of the same work, stored on various conventional devices, but no way to keep track of where the most recent version resides. Nomad does not however restrict itself to personal computing environments, but is also intended to be an information sharing system in much larger contexts, such as project groups. Information gathering is only performed on request from a user (pull-based-system), while information sharing on any of the interacting devices is always active, as long as that device remains on and connected to the network. [21]

1.1.1. Project History and Status

Nomad was conceived in June 2002, and begun without funding in that year. It became a Microsoft/THRIP funded project in 2003. In 2004, the funding was diverted to RoSCtor. In 2005, the project was started again in earnest, with the interest of ELogics. At the time of writing, this work is supported by Microsoft, THRIP and Elogics. At the time of writing, the first release of Nomad was being developed. It aims to implement the basic framework, with extensibility being a main concern.

1.2. Problem Statement

This section provides a statement of the problem that this thesis intends to address.
The intention of this thesis spans from the design of the protocol with which Nomad intends to handle disconnectivity and explore how the protocol may be improved to handle a variety of scenarios. We show how the protocol will set up the community

via email, and maintain the distributed artifacts among the nodes. Nomad is intended for a small closed community where members randomly or voluntarily, switch from being online to offline. Nodes may be intended to go offline, due to working hours of its user, or randomly due to a weak internet connection. The Nomad protocol takes this into account and has the ability to handle such cases.

The proof of concept that this thesis bases its findings on has been developed in a way which allows full customization of individual nodes. We aim to show 3 specific types of nodes, namely, a workstation, a mobile laptop, and a PDA. Other node types can be easily added. The characteristics of these nodes are shown in section 3.4. The simulation will show how Nomad will be able to customize different reactions based on the type of the node. Nomad is primarily used as a *"best-effort-approach"* to gathering the most updated artifacts. Further emphasis will be placed on how Nomad will be able to handle situations where nodes that are never online at the same time. This could be it due to a variety of reasons, possibly from time zone differences, and hence different working hours. These nodes will therefore never be able to gather information from each directly. The ability of another node within a community to act as a propagation node will resolve this problem. We investigate the advantages and disadvantages of this approach.

1.3. Motivation

This section offers motivation of why this problem is important and why a solution will be interesting.

Nomad is intended to be a CSCW tool. Nomad has been in the design phase for most part of its life cycle thus far. The main reason emphasis has been placed on the design, is due to the fact that Nomad should be made robust enough to handle current and future technologies as well as be a usable tool for nomadic workers. In recent years, enabling technologies such as the Microsoft .Net framework have allowed ease of development, the ability to develop for multiple devices with a variety of connectivity options and security aspects. But with the start of new technologies, arise the problems of uncertainty of how these technologies will react under a variety of situations. Furthermore, the Nomad requirements have changed over time to include a variety of situations to which users have become accustomed. Simple examples are

ease of use and the ability for the application to run transparent to the user, with as little user intervention as possible.

A simulated environment has the ability to show a number of situations under controlled conditions. For this reason, it would be of utmost interest to evaluate how the Nomad protocol will perform under a variety of environments. This will have a direct impact on how Nomad will be expanded for future use. It may be shown that certain objectives that Nomad intends to achieve may be overkill and might even have a negative effect on the system.

1.4. The Need for a Simulator

The Nomad Protocol Simulator (NPS) has been developed and coded in parallel with the first release of the actual Nomad application. The initial version of the Nomad application will have the basic framework implemented, with hooks to include later *pluggable* add-ons and new features. Once the initial version has been released, the results gathered from this thesis will be consulted for development of future releases of Nomad application. It is therefore necessary for this thesis to produce "*close to real life*" results and iron out problems that Nomad might encounter in certain scenarios.

1.5. Contributions of this thesis

The intention of this thesis is to illustrate scenarios that Nomad will encounter, and how the underlying protocol intends to realize each of these scenarios, with specific emphasis on the connectivity aspect of the protocol. It is not intended to show Nomad as the total application, since Nomad itself has many components. We do however take a broad look at aspects that had a direct influence to the design of the NPS as well as the perspective to which Nomad would fit into the current world. Some of these topics include distributed systems, Computer Supported Cooperative Work (CSCW) and peer to peer systems. This thesis intends to offer valuable insight and reasons that led to use of a simulator, as well as outline the boundaries of the protocol and provide a possible solution for Nomad. The Nomad Protocol Simulator (NPS) intends to be a useful tool for determining outcomes from possible scenarios. The results from the simulator will be taken into consideration for future releases of Nomad. We do not place any emphasis on the security aspect, but it is discussed in the future work section of this thesis.

1.6. Document Roadmap

This section provides a brief overview of the structure of this thesis. There are 8 chapters in total.

Chapter 1 provides a background to Nomad. We state and motivate the problem that this text aims to address. In chapter 2, we take a broad look at central concepts relevant in Nomad. These serve as background knowledge of this text. We compare these concepts to related work and conclude by demonstrating uses of Nomad. In chapter 3, we explore the proof of concept that this thesis presents. We show how the Nomad Protocol Simulator (NPS), simulates aspects related to Nomad. We define design considerations and assumptions made by the NPS. Chapter 4 presents the scenarios developed for this text. The scenarios aim to provide a better insight to the potential problems faced by Nomad, as well as provide a possible solution. We discuss shortfalls, advantages and disadvantages of the intended protocol to be implemented by Nomad. We provide possible solutions and highlight possible pitfalls of the protocol. The scenarios illustrate concepts discussed and presented in previous chapters of this text. Much work has been done with regard to the possibility of the integration of mobile agents in Nomad. In chapter 5, we discuss and motivate this concept. We show advantages and disadvantages of the use of mobile agents in Nomad. We conclude this chapter with reasons of why this concept will not be implemented in the first release of Nomad, and hence was not included in the NPS. Chapter 6 evaluates Nomad against other related Computer Supported Cooperative Work (CSCW) systems as well as similar mobile agent systems. We discuss the differences and similarities relating to these projects. This chapter also takes a look at features of the .Net framework of interest to Nomad. The .Net framework is the enabling technology that the NPS has been developed on and the framework that Nomad is currently being developed on. The concluding chapter, chapter 7, sums up the main findings of this text. We look at possible future work, extensions and improvements of Nomad and the NPS.

2. Background and Related work

2.1. Central Concepts

2.1.1. Distributed Systems

In order to distinguish what a distributed system is, we need to understand what it is not. Table 1, tabulated from Emmerich [38], differentiates a centralized system to that of a distributed system.

Centralized system	Distributed system
Has non-autonomous components	Has autonomous components
Often built on homogeneous technology	Often built on heterogeneous technology
Multiple users share resources of a centralized system at any time	A single user allows exclusive access to unshared components
There is only a single point of control, and therefore, a single point of failure	There are multiple points of control, and therefore, a multiple points of failure

Table 1 : Characteristics of centralized and distributed systems

Emmerich [38] states that distributed systems aim to provide the following non-functional requirements:

- Scalability
- Openness
- Heterogeneity
- Resource sharing
- Fault tolerance

Nomad has been in the design phase for most part of its lifecycle. We now take a look at how Nomad meets, or intends to meet the non-functional requirements of a distributed system.

Scalability denotes the ability of a system to accommodate growth, be it expected or not. Nomad intends to host a small community. The NPS has shown that the protocol allows for scalability of a small community in terms of number of users. For the purposes of this thesis, we assume that the number of nodes in the small community is a maximum of 10 nodes, and the average community is between 3 and 8 nodes. This by no means suggests that the protocol will be unable to handle a larger number of nodes. Nomad has a decentralized architecture, and the design accommodates for a *server-less* environment. This allows for concurrent users to share their processing and data load. In terms of *geographic scalability*, Nomad uses the internet and email as its backbone. Therefore, Nomad can scale geographically, as long as there is a valid internet connection.

Openness denotes the ability to easily modify and extend the system. The intention of Nomad is to have an initial basic framework, and later add extensions as needed. An intended extension in the near future is to add mobile agents. This is further discussed in section 5.

Heterogeneity of components arises from the use of different technologies within the system. This could be caused by the use of a variety of programming languages, operating systems, hardware platforms and network protocols. Nomad allows for a variety of hardware components. These span from, but not exclusive of, PDA's, workstations and laptops. Current development on Nomad is undertaken on the Microsoft .Net framework. .Net facilitates for well known protocols, variety of devices, and multiple programming language support [30]. Mono is the .Net equivalent for the Linux operating system. For these reasons, .Net has been chosen as the development tool for Nomad.

Hardware, software and data form part of *resources* in a distributed system. In Nomad, a propagation node is a shared resource. This node would be accessed under certain scenarios as described in section 3.1.2.3. Each node has particular user-defined sections, which store the artifacts per project. Security therefore becomes an aspect for Nomad. Although security is not covered in this thesis, it is planned for future work.

Fault tolerance of a system demands that operations continue, even in the presence of faults. Nomad has the ability to continue even if a particular node is offline, or not available, and hence not in the community. Error prone connections, due to weak connections are handled as well. Replication in Nomad *could* be facilitated by a propagation node. Peer to peer systems are generally fault tolerant. Section 2.1.4 discusses how Nomad relates to the P2P architecture.

Along with the above requirements, it is often the case that designers of distributed systems hide the complexities of the distributed components from the end user. This is termed *transparency*. Nomad intends to have minimum user interaction. If the user invokes Nomad, Nomad would create connections to other nodes, gather the needed artifacts, queue requests if needed, integrate the newer updates into the project, and update possible dependencies. All these operations would be transparent to the user, and independent of the location.

2.1.2. Computer Supported Cooperative Work (CSCW)

Computer Supported Cooperative Work *(CSCW)* is the study of how people use technology, with relation to hardware and software, to work together in shared time and space. CSCW began as an effort by technologists to learn from anyone whom could help better understand group activity and how one could use technology to support people in their work. These specialists ranged from vast areas of research, which spanned, but not exclusive of, economists, social psychologists, anthropologists, organizational theorists and educators. [16]

Technology already plays an important aspect in our everyday lives. From the advent of the first telephone, to the current usage of email, cellular phones and instant messaging, humans continue to be social creatures, who aim to keep in touch, whenever and wherever. In fact, emails and cellular phones are tools of CSCW. In addition, it was found that instant messaging [7] is a CSCW tool. A more recent tool which has made an impact in the social arena of CSCW is *blogging* [8]. Blogging is a web based communication tool which allows individuals, small groups with a limited audience to share information. Personal views are placed in the commonplace of worldwide criticism. The aim of these systems is to give us the ability to collaborate and communicate at will. Each system is used under different scenarios.

There are two dimensions of CSCW. These are *space* and *time*. Figure 1 shows examples relevant to this study in the usual CSCW quadrants..

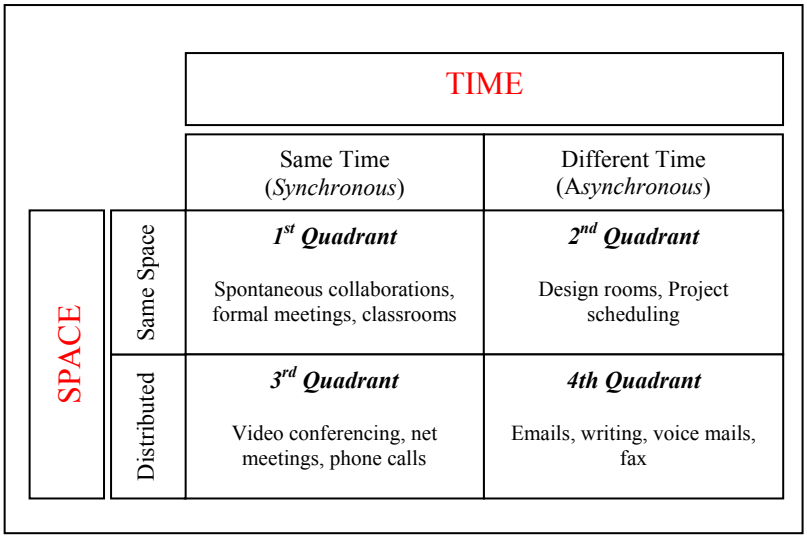


Figure 1 : CSCW Quadrants

We illustrate these concepts by means of the following example. Assume David and Andrei are working on the same project. They are collaborators in a project.

If David and Andrei need to share information *immediately*, they are sharing their time. This is termed *synchronous*. An example of this is the cellular phone. Although the collaborators are not in the same place, each individual is connected by the time that they share for the duration of the phone call. This example would fall in the 3rd CSCW quadrant.

If David sends Andrei an email, they are not sharing time. This is termed *asynchronous.* Andrei will only have the information once he reads the email. This example would fall in the 4th CSCW quadrant.

In the examples above, David and Andrei are not in the same place when the information is shared. They are therefore *distributed*. The following examples show *synchronous* and a*synchronous* situations where the collaborators are in the *same place*.

Information sharing can take place when David and Andrei meet for a cup of tea at tea time to discuss the project. Although this is a spontaneous collaboration, information is still shared between the two collaborators. They share the *same place*, and share the *same time*. This example would fall in the 1st CSCW quadrant.

David and Andrei occasionally visit the company's intranet blog, where other team members share their ideas, new found bugs, bug fixes and see the general status of the projects they work on. They hence share the *same place*, but visit the web-blog at different times. The information shared in this instance need not be shared immediately and is generally formed over time, but is still relevant to the totality of the project. This example would fall in the 2nd CSCW quadrant.

Nomad falls in the in the 4th CSCW quadrant. Nomad aims to provide collaborators which do not share *time* or *space* to easily collaborate and communicate as a *virtual team* and co-exist in a *community*. The terms *virtual team* and *community* is discussed in section 2.1.5 of this text. MOTION ([14],[15]) is a comparable project, similar to Nomad, which supports the collaboration of nomadic workers. Other projects which are considered CSCW based are Novell iFolder [48], SubethaEdit [50], Microsoft Sharepoint [44], CoCoDoc [13], Basic Support for Collaborative Work (BSCW) ([4],[5],[6],[7]) and X-peers [52]. These systems are compared directly with Nomad in section 6.1.

CSCW is a multidisciplinary, which has gained the interest from many fields [16]. For example, in the 2004 CSCW conference [9], papers were received from the medical, social, gaming, organizational and computer disciplines. From a broad view, one can say that each discipline showed the effect that CSCW has on their everyday collaborations, and how communication could be improved between collaborators. In a recent paper [10], the authors look at the evolution of the CSCW research from its inception to 2004. Among their discoveries, they noted that over time, new CSCW members entered the community, along with old members playing an influential role in the CSCW community. New members could play an influence in defining CSCW and where it is heading, although core members would not allow it too stray to far from the original intentions. The challenge of CSCW being multidisciplinary is that a single word, used in a different context, has a different meaning for different groups

with different global background. A simple word could be interpreted differently by an individual. Each individual has different priorities, and hence a different perspective of CSCW. One might oversimplify or over generalize, and in doing so, might overlook important distinctions. An example could be the word *"implementation"*. To a developer, *implementation* could denote a very detailed coded algorithm. To an information systems manager, *implementation* might mean the introduction of a new system to a company. No one assumes that everyone is expected to talk in the same language, but we could help in the way that we communicate with each other. We note what people have to offer and decide if is in our interest to acknowledge it. [16]

Although CSCW and Groupware are difficult to define, and no single definition satisfies everyone, there are individual conferences held for each. CSCW is used to define the research and Groupware defines the technology [16]. It is also common to differentiate between these terms so that Groupware is technologically focused and narrows the social forms of cooperation that CSCW spans. [23]

For the purposes of this text, we make a distinction between CSCW and Groupware. Groupware is considered the enabling technology, be it hardware, software, services and/or techniques, which allows people to work in groups. CSCW, on the other hand, focuses on the *study* of tools and techniques of groupware as well as their psychological, social, and organizational impact. We now discuss Groupware, and aspects relating to Groupware.

2.1.3. Groupware

Groupware also known as *collaborative software* [23], allows many concurrent users to work on the same project. Whereas a single user system focuses on the individual, Groupware focuses on the group. What advantages does a Groupware system offer when compared to a single user system? When working on a project where communication is essential between collaborators, Groupware facilitates communication faster and clearer, and enables communication where it would not otherwise be possible. It aims to allow multiple perspectives, expertise and assistance with group problem solving. It aims to save time and cost in coordinating group work.

With this in mind, Groupware is far more complex to design and maintain than a single user system.

As stated earlier, blogging is a tool for CSCW. Blogging itself is not defined as groupware, but the blogging system and infrastructure which allow groups to communicate and collaborate via web pages. It is possible to maintain different identities, comment on other people's views and allow collaborators to manage and coordinate multiple posts, or projects. These features qualify blogging as a group support technology. The general approach of groupware, as will be seen with each of the systems compared in section 6.1, is to allow distributed members in a group to collaborate through some type of infrastructure. The type of infrastructure is dependant on the service the system aims to offer.

Design

Groupware is generally created around the users of the system. Understanding of the context of how the system will be used within the group is essential. Analysis of the type of users and their intentional use of the system is crucial. The system is being created for the user. The user has to accept the system and group has to adopt use of the system. For example; users might refuse to use the system if the system is not easy to use or understand. This would result in a failed system. Prinz [39] confirms the changing nature of groupware. Requirements pose a problem when they are changed over time, and much time can be spent on formal processes. Prinz abandons these formal processes and opts for using scenario based understanding; rapid prototyping and user driven priorities. Similar problems were faced with Nomad. Nomad remained in the design phase due to changing requirements, deeper understanding of the user's needs and assumptions of outcomes based only on "*gut feel*". In this respect, the design and creation of the Nomad Protocol Simulator (NPS) aimed to clear idealistic views of Nomad, show possible pitfalls and possible improvements to the protocol. Scenario based events form a core part of the NPS. The questions of *"What if"* and *"is this possible"* are not only answered, but evidently proved. We now consider some design aspects relating to Nomad and its users.

In Nomad, collaborators are content to receive an artifact which might not be the latest version of the artifact. The collaborators would like a global view of the status

of the project, or status of only part of the project. The *best effort* approach to collecting artifacts is sufficient, although this text offers alternate approaches for certain scenarios discussed in section 4.3.

Communities are made up of a number of nodes. The smallest community will have two nodes. And since it is expected that the system has a 100% uptime, Nomad should handle the special case where both nodes are never online at the same time. Communities are discussed in section 2.1.5.

Users will be mobile for most part of the project lifetime. They could connect to the community for short periods of time, using any available internet connection. Nomad thus has to handle a variety of heterogeneous internet connectivity protocols used by a variety of devices. We look at possible internet connections that the NPS assumes in section 3.4.

Although there are many aspects relating to Groupware, the design of Nomad was given special attention. Particular care had to be made regarding transparency and ease of use. The initial release of Nomad is intended only as a proof-of-concept. Once this can be demonstrated, Nomad will continue to blossom and new features and *"nice-to-haves"* will be added.

2.1.4. Peer-to-Peer (P2P)

Peer to peer (P2P) networks have gained acceptance and their applications are synonyms with file sharing. More focus though has been brought onto P2P networks due to the legality struggle with the Napster architecture in the recent years. With this, many new P2P networks have spawned off, including the Gnutella architecture which, unlike the Napster centralized architecture, is based on a decentralized architecture. Androutsellis-Theotokis [33] has prepared a detailed analysis of these systems, and a wide range of other P2P systems. Iamnitchi [2] states that P2P networks are preferred due to their ability to operate robustly in dynamic environments. Furthermore Cugola [15] states that uncoupling from the client-server architecture is better for mobility since it frees clients from coupling with servers and allows an arbitrarily large and continuously changing set of nodes to be accessed at

once. Therefore the search space is dispersed over the nodes. We take a broad look at these types of systems in this section.

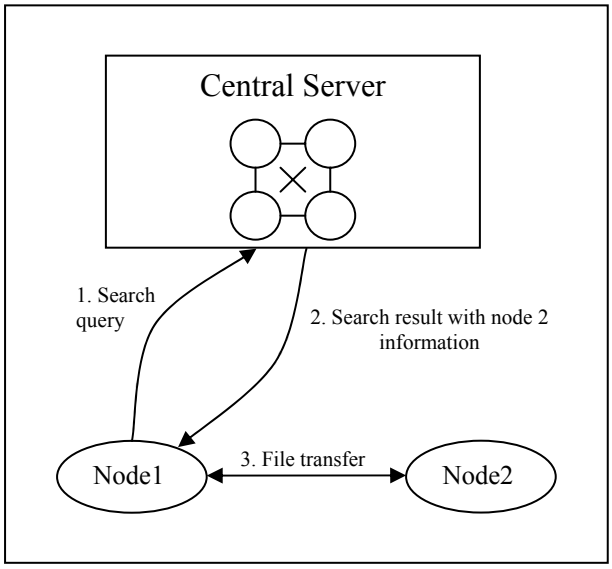


Figure 2 : Centralized file transfer

A centralized architecture normally consists of one or more centralized server(s) which contains indexes of files that exist on each node in the network. Figure 2 illustrates the idea behind file downloading in a centralized architecture. The requestor node, *Node1*, queries the central server. The server replies with a search response. If a match for the file is found, information of the node that has the requested file is sent with the search result. The requestor node then transfers the file from the node that has the file. Lan[18] states that the centralized architecture suffers from two limitations. Firstly, the indexing server could be a bottleneck and single point of failure. Secondly, since the indexing information on the server is refreshed periodically; the indexing server could return stale information to requesting peers. The files could have been deleted at a peer, but since the server does not acknowledge this until a refresh, stale information is still sent.

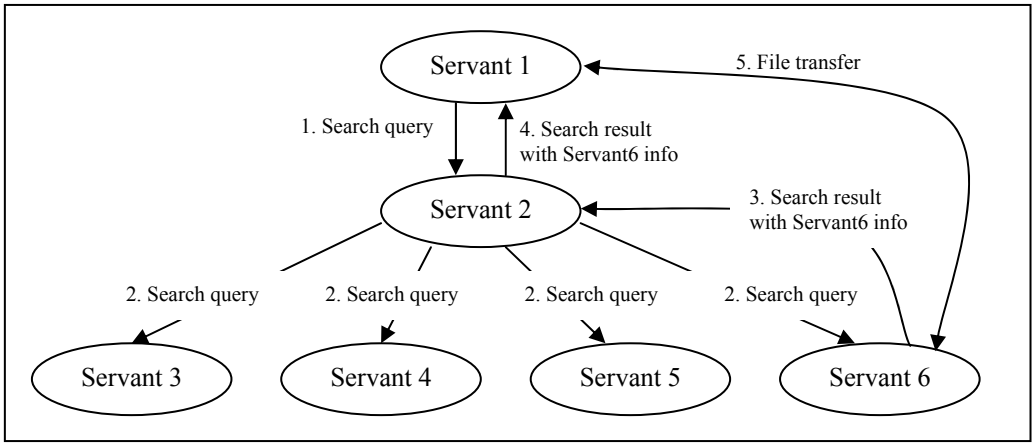


Figure 3 : Decentralized file transfer

In a decentralized P2P system, all peers are the same, except for the files they share. Each node acts as a client and server. This is termed *servant* [51]. There is no single point of failure, nor is quality of the network compromised if a node is removed. There is no global index and no central coordination. Figure 3 illustrates the idea behind a file transfer in a decentralized architecture. Servant 1 is the requestor. Servant 1 queries the nodes closest to it, in this case Servant 2. Servant 2 then queries all the nodes that it is close to, Servants 3-6. Search requests typically have a time to live (TTL) counter [33], which decrements on each hop. Once this value reaches zero, the message is dropped. This normally determines the depth of the search. Since Servant 6 has the requested file, it returns with the search result along with its info. This information contains the IP address, port, node transfer speed and number of file matches.

Overlay networks combine centralized aspects into decentralized architectures [33]. Certain nodes assume a more important role and are aptly named *super nodes*. These nodes are dynamically assigned, so failure in a super node will not cause a failure, but merely a newly assigned super node.

Nomad tends to have a single depth search, since each node is aware of the other node, yet there is no central server maintaining the indexes of each node. The number of nodes in the community is small, so this approach is feasible.

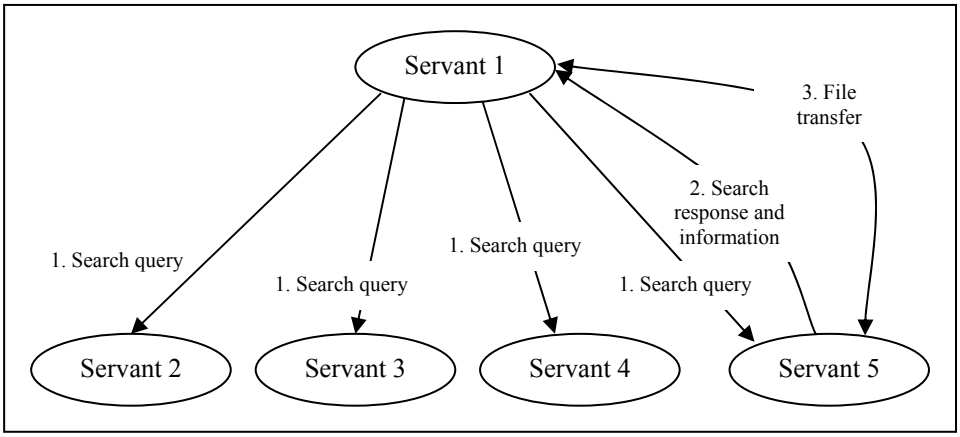


Figure 4 : Nomad file transfer

Figure 4 illustrates the idea behind a file transfer in Nomad. The requesting node, Servant 1, sends a request to all online nodes. If a search is successful, a search response is sent back along with nodes information. Special cases exist where requests are made while certain nodes are offline. The files or requests are sent to propagation nodes. The propagation node adds a centralized aspect to the architecture, and like overlay networks, can be dynamically assigned. Propagation nodes are discussed in section 3.1.2.3 of this text. Cugola [15] states that most current P2P based file sharing applications are content with a *best-effort* approach. An empty result set is acceptable, even though the file is within the P2P network. This is not acceptable in enterprise applications. Nomad is not intended to be an enterprise application, so the best effort is acceptable. If a user requires an artifact from a node that is offline at the time of the request, Nomad has the ability to change its effort levels to allow a requestor to receive the required file via a propagation node. Desired effort levels are discussed in section 4.3 of this text.

Failures are inevitable. In an individual system, a hard-drive could crash; one could suffer from a network failure or a loss of power. In the case of P2P, failures could be a bit more complex [29]. Designers of P2P systems should be able to anticipate and handle software failures, partial or total communication interruption and users who join and leave the community independently at will, regardless of time. Failures are bound to happen, but the system should be able to tolerate such failures and more importantly, recover from such failures.

Much work has been done with respect to P2P systems. These vary from improvements to the protocol [42], considerable measurements over time ([17],[32]), maintenance in the network [18] and comparison of search methods [36]. The NPS intends to show the protocol in terms of events. The running simulation time of these simulations is mostly under 20 minutes. The intention is to show that the intended protocol will sufficiently maintain the nodes and allow them to interact.

2.1.5. Communities

A community or *virtual community* is defined as a set of members that share a common goal. Communities are synonymous with small world networks ([1],[2]). The goal of a community can be broken into individual projects. From the set of community members, a subset works on each project. A high level diagram depicting the notion of a community is shown in Figure 5. Individual members may overlap between projects. Similar definitions are defined by Cugola [15] and Reif [14] for the MOTION project. Khambatti [24] defines *seers* as being more influential peers. Nomad defines every peer to be the same. All peers are on the same level.

A set of members that belong to a project are known as a *virtual team.* A virtual team in Nomad consists of one *project initiator* and many *project collaborators*. The project initiator acts like any other collaborator once it has sent the initial email to all the collaborators. This is further discussed in section 3.3.1.

The members share common artifacts related to the project. Since this community exists in the realms of cyberspace, they could have an *online* or *offline* status. The community could therefore have a subset of online community members. We therefore have a *dynamic* online community.

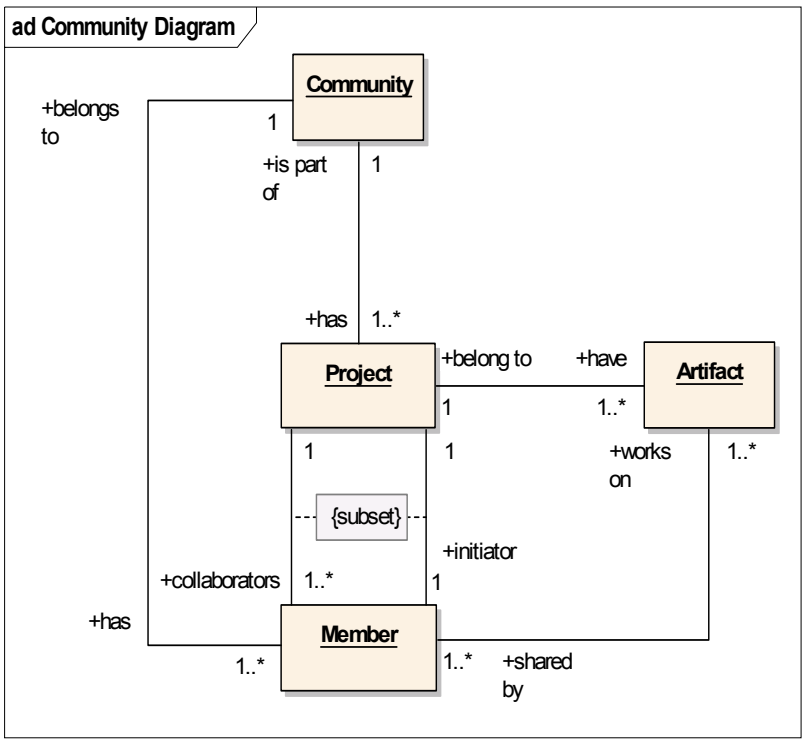


Figure 5 : High level community diagram

One community can have many projects. Each project has many members and artifacts. The artifacts that belong to the project are worked on and modified by the members of the project.

A community has a cause and purpose, striving for a common goal. For example, the goal could be to author a book. This community could have a single project. Each member is a specialist in their field. Each member could be in charge of individual chapters. Each of these chapters could be an artifact. If for example, the chapters were too high level and each specialist worked on sections in the chapter, then each section could relate to a single artifact. These artifacts could be code snippets, or pictures, in fact this paragraph could be a single artifact. The granularity of an artifact and project are choice driven by the intended members of the community. The artifacts therefore vary in size. Artifacts within the project could have dependencies. For example, if a code snippet was modified, the code output and related diagram could have been modified. The code output and related diagram are therefore dependencies of the code snippet. For the purposes of this thesis, artifact dependencies are not included. Artifacts are further discussed in section 3.1.2.2.

2.1.6. Push and Pull Technologies

Cheverst [22] distinguishes the notion of *push* and *pull* as the intention of the user and the dissemination of information. Information *pull* is defined as any information that occurs due to a conscious initiation by the user. Information *push*, on the other hand, is information that arrives at the user unexpectedly, but normally due to some subscription to a service.

Hauswirth [25] contrasts the pull and push models as illustrated in Figure 6.

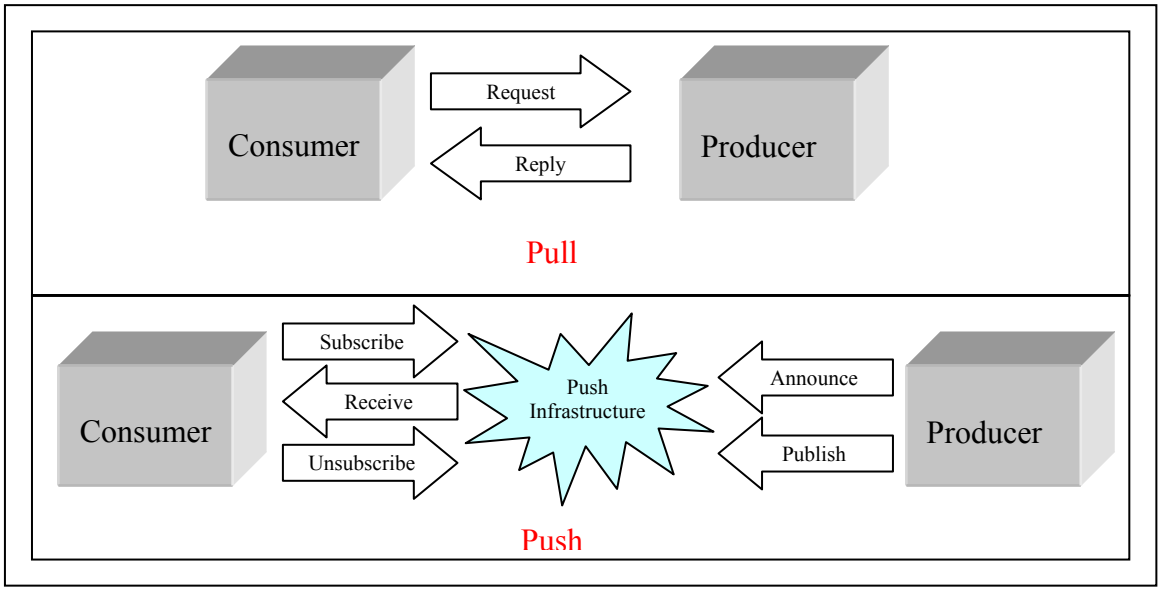


Figure 6 : Push vs. Pull model

The most common example of a pull system is web browser. Information is *pulled* off the server when the website is accessed and delivered to your web browser. The information would have not been accessed, nor would it have arrived at your web browser if the website had not been accessed. The information was expected by the user.

In a push based system, the user usually *subscribes* to a service to which updates are received without the user having intended to receive this information. A common example of this is an email news subscription. The user subscribes to the service and thereafter receives email on a regular basis when it is published from the service. The email is therefore *pushed* to the user. The infrastructure in this instance is the email system. The user has the option to unsubscribe to the service to stop these messages. In the case of Nomad, a user would unsubscribe from being a member of a project.

Nomad is intended to be a *pull* based system. Artifacts are sent to the user only on demand. But Nomad does have *push* characteristics which make it both a push and pull based system.

Nomad *pulls* artifacts from available online nodes when needed on a best effort approach. But what happens if the node requires an item from a node which is never online at the same time as it is? The node therefore resorts to using a propagation node. It should be noted that there can be more than one propagation node within the community. We discuss the flow of execution in the following example.

Suppose the two collaborators, Vishani and Meeshaal are never online at the same time. Vishani requires an item from Meeshaal, but Meeshaal is never online when Vishani is, due to working hours and time zone differences. This example assumes one propagation node in the community. After a decided number of failed attempts, set in the NPS as five, Vishani's request is by design *pushed* onto the propagation node. If there are more then one propagation node, the request is forwarded to *all* available propagation nodes in the community. The propagation node attempts to send the request to Meeshaal's node, assuming that Vishani's node might have the incorrect IP address for Meeshaal's node. After a desired number of attempts, the propagation node waits until Meeshaal's node becomes available. Generally the propagation node attempts a single time before assuming that Meeshaal is offline. When Meeshaal's node is available, the propagation node *pulls* the required item from Meeshaal's node, mimicking a request from Vishani's node and stores the required item until such time that Vishani's node is back online. When Vishani returns to work the next day, she becomes available to the community. The propagation node then *pushes* a notification to Vishani stating that the stored item is available. The reason that this is done is that there might be more than one propagation node in the community and therefore possibly multiple notifications that are returned to Vishani's node. Assuming the item is an artifact; Nomad selects the best artifact based on the most updated version, and downloads the artifact from the selected propagation node. If there is more than one possible selection and if user interaction is on, the Nomad system prompts the user, Vishani, to choose her preferred provider. If however, there is only one notification that is received, Vishani's node will automatically *pull* the

artifact from the propagation node. If an update has been made, the user is notified. This scenario is further discussed in section 3.1.2.3.

Another case where Nomad *pushes* information to its users is when an *online notification* is sent. Any Nomad collaborator *subscribes* to *online notifications* and emails implicitly when they become members of that project. Section 3.3.2.2 discusses this further in detail. A similar approach is followed by the MOTION [14] project.

2.1.7. Connectivity and Casual Connections

If *connectivity* is the process of having direct access to a network, and being available to the community, then *disconnectivity* is the process of disrupting the access to the network and not being available to the community.

Local Area Network (LAN) connections have a *strong* connectivity whereas a wireless connection is considered a *weak* connection. The reason for this is quite apparent. A LAN connection has a *less* probability of having its connection disrupted, although it is possible.

A fundamental problem in wireless ad-hoc networks is connectivity [3]. An ad-hoc network needs to be well connected to be useful. A wireless connection has a *higher* probability of having its connection disrupted. Examples of disrupted connectivity are loss of signal or weak signal, battery loss, network coverage or moving out of an access point range.

Reif [14] states that a system needs to have three modes of connectivity for nomadic working. These modes are as follows:

- *Connected mode*: connectivity is strong and a direct connection to the Internet is available from a fixed network node.
- *Disconnected mode*: the user has no network connectivity. The user can still continue to work even though the user has no network connectivity.
- *Ad-hoc mode*: the user creates a spontaneous network between devices that is weak and not optimal. None the less, collaboration still takes place.

A *local repository* is typically updated while the user is disconnected and the new version is updated in a version control system when the user connects once again. There is no need for Nomad to update any other central repository when it becomes online again.

For the purposes of this text, we define *intentional* and *unintentional* disconnections. An intentional disconnection is a disconnection that is expected by the user. A user disconnecting when going home after a day's work is an example of an intentional disconnection. An unintentional disconnection is a disconnection that is not expected by the user. The disconnection could be due to a weak network connection or a sudden loss of power. For the purposes of this text, we state that an unintentional disconnection is merely a special case of an intentional disconnection.

Tang [3] realizes a formula that calculates the probability that an ad-hoc network is fully connected. An ad-hoc network would be most optimal when fully connected. Nomad spans to host a variety of devices. We aim to provide the users with ways to share files regardless of how connected or disconnected the community is. The files are shared on a best effort approach, with an option to use a propagation node if a user is not connected for a long enough period of time to pull the needed artifact from another offline node.

The availability of hosts in a P2P system is a function of time [29]. Availability could simply be stated as the host being online and accessible by the community. Bhagwan [29] continues to state that measuring host availability can be misleading and instead the unique ID of a host should be used instead. This approach is followed by the NPS. Furthermore, in the simulation carried out on Overnet, a live P2P network, each host entered and left the system 6.4 times a day on average, and a total of 20% of the hosts arrive and depart every day [29]. These measurements were taken over a period of seven days. These values are interesting since it shows that users vary their availability due to time of day.

Users who access the community at will and at a variety of times could be dubbed as *casually connected* users. They do not have a permanent connection to the community, and are made available at their own discretion.

The concept of time difference could be due to users being globally dispersed and in different time zones. Time therefore plays an important factor in Nomad. In section 2.2 we note some of the uses of Nomad. Time is a significant factor in each of these cases.

2.2. Uses of Nomad

We now take a look at scenarios where Nomad can be used. These examples were first noted by the members of the Nomad development team [21].

2.2.1. Authoring

Overview

This is a classic example of two authors working on the same book, article, document or proposal. These "*specialists*" work on a single deliverable. They are distributed worldwide and are highly mobile. Each author is responsible for parts of a deliverable, and need to view other authors work to ensure that the deliverable forms a unity.

Why Nomad?

- There is no guarantee as to the connection status or availability of each worker.
- A best effort view of the document is acceptable. The workers do not need the latest, complete version of the deliverable.
- The option exists to allow the latest copy of the deliverable if needed through the use of alternate effort levels.
- The specialists *roam* between multiple connection points and use any available internet connection point to collaborate.

2.2.2. Student Administration

Overview

In a complex environment, such as a university, multiple lecturers and tutors are responsible for providing marks and comments on a student's progress. University management has received complaints from students that the marking of student deliverables, such as assignments and projects, takes too long. Management would

like to view the progress of marking at any point in time. This would make it easy to identify any bottlenecks in the marking process. Each marker is required to enter these marks into a spreadsheet upon completion of marking.

Why Nomad?

- Because of the tedious nature of marking other peoples work, the markers might decide to work in coffee shops. They will therefore only take their PDA's along to enter the marks and update via a wireless link offered by the coffee shop.
- Some markers, especially the tutors, do not have constant internet connections. They might choose to go online once or twice daily.
- Management would like to be able to draw a mark sheet *at any given time* which will show the latest available marks.

2.2.3. Route management

Overview

A parcel delivery service company would like to be able to draw reports *at any given time* of the day to show the progress of deliveries and pickups. These reports can be used to re-route drivers to maximize throughput. The company makes use of freelance drivers and therefore are not allowed to install tracking systems in their vehicles. When a driver makes a stop, they are required to capture the client's signature on their PDA. The mobile device will then give instructions to the next stop.

Why Nomad?

- Drivers can only connect to the internet via a wireless internet connection, such as GPRS. They might roam into areas that do not have internet connectivity.
- Management requires a best effort view of the progress from all drivers.
- Drivers use mobile devices to capture and store their own progress.

2.2.4. Scientific research project

Overview

A non profit scientific research organization is doing research on the behavioral trends of chimpanzees and would like to improve the collaboration between scientists. Some

of the scientists are situated in the remote areas of the African bush while others are pursuing research in high tech laboratories. The financial support organization would like to be able to view the current state of the deliverables at any given time.

Why Nomad?

- Researchers make use of mobile devices to capture and store data.
- Scientists work together to compile reports.
- We can introduce a data mining server, which use the results gathered from the researches to identify trends. These reports will then be viewed, updated and validated by the field researches. The server could act as a propagation node in this case.
- Different media types will be shared. Field researchers might capture the chimps on video while lab researchers might draw graphs and write long boring reports. Each type simply represents an artifact.

2.2.5. Software development

Overview

Many software development companies want to improve their efficiency by working a 24 hour day. This can be achieved by employing coders worldwide in different time zones. Multiple coders will work on the same software component. This concept has been realized in open source development movement. An example of such a system is *Sourceforge* [49]. Open source projects usually have a controlling body or company that is responsible for the requirements, quality and support of the software project. The real power lies in the fact that potentially thousands of programmers solve the same problem differently and the controlling bodies have the freedom to choose the implementation they desire.

Why Nomad?

- Coders are distributed not only by space, but also by time.
- Nomad will assist the controlling bodies in retrieving the best available version of artifacts based on their desired criteria.
- Coders may not be online constantly.

2.3. Conclusion

This chapter has introduced central concepts relevant to the understanding of the rest of this thesis. We have shown Nomad to be a CSCW Groupware tool that allows distributed members in a small community to collaborate via technologies such as P2P. We discussed the design decisions for Nomad and highlight the use of the NPS.

We illustrate examples of how Nomad will allow collaboration. We continued to show uses of Nomad and why Nomad would be useful in particular scenarios.

The chapter that follows introduces the NPS and design decisions regarding the NPS and the Nomad protocol.

3. The Nomad Protocol and the Simulator

3.1. The Nomad Protocol Simulator (NPS)

This section discusses the proof of concept for this thesis, the Nomad Protocol Simulator (NPS). A high level class diagram is shown in Figure 7.

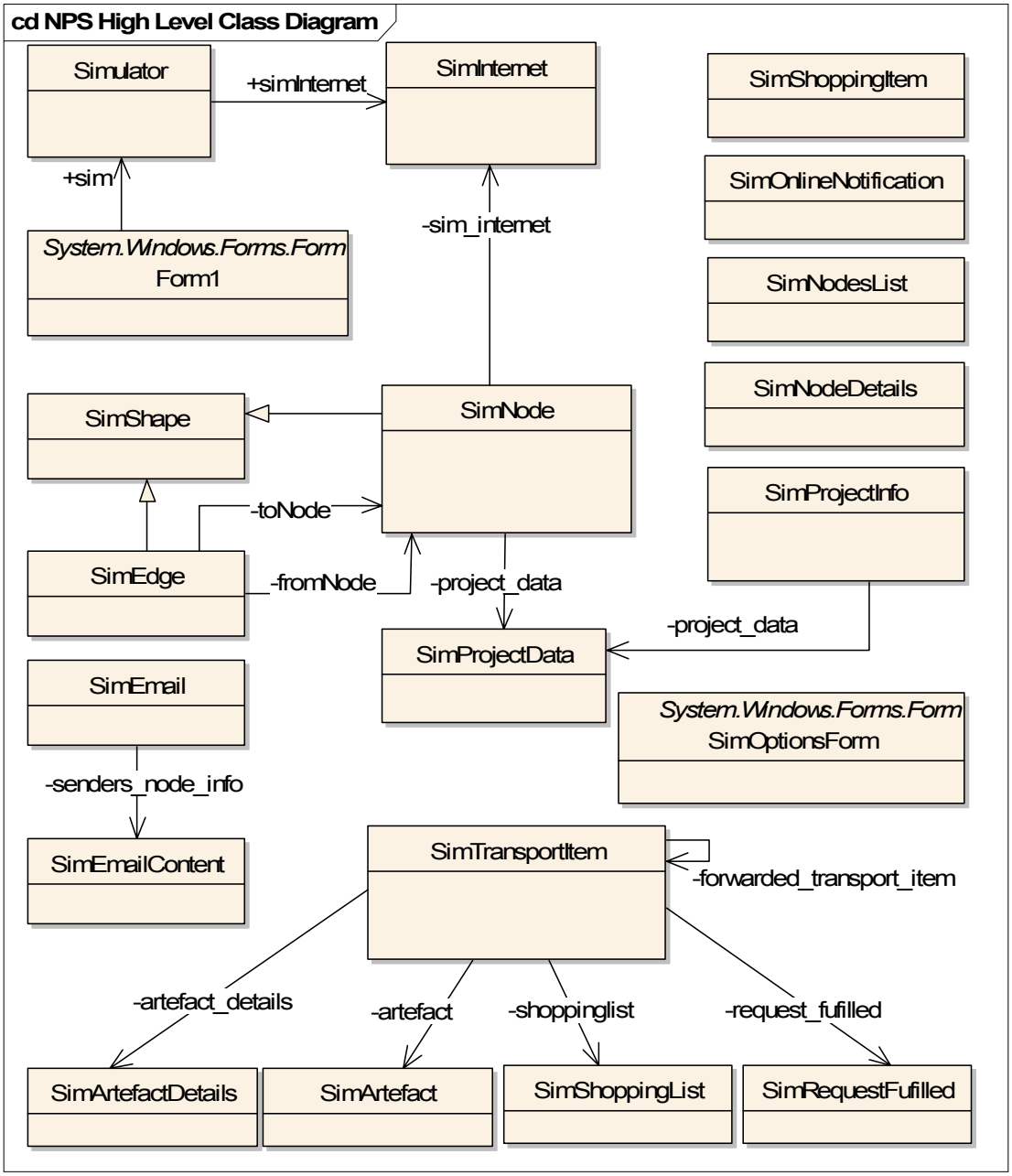


Figure 7 : NPS high level class diagram

The NPS has been developed with the .Net version 1.1 Framework. Nomad is being developed in the .Net version 2.0 Framework. The reason for this is due to the fact that version 2.0 offers a wide range of new technologies that could easily be implemented along with Nomad functionalities. We discuss some of these features in section 6.3.1.3 of this text.

3.1.1. Interface

We take a brief look at the NPS user interface. This will create an understanding of some of the terms later used in this text.

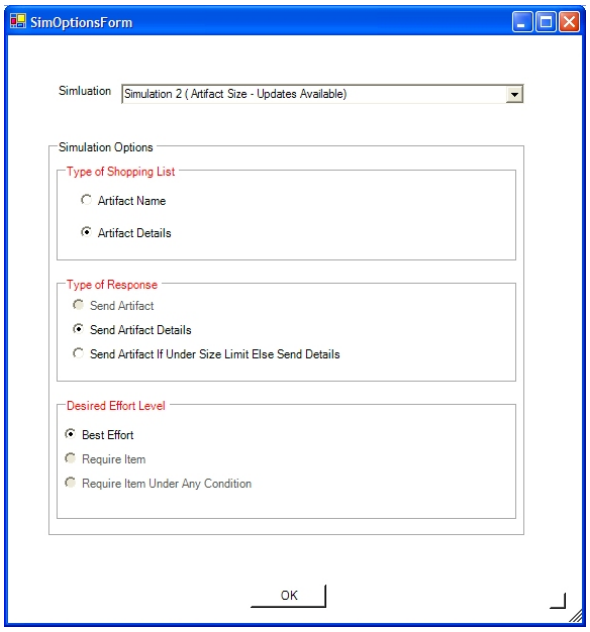


Figure 8 : Simulator options form

When the NPS is initiated, a windows form as in Figure 8 is shown. The form shows the available scenarios. With each available scenario, relative choices are shown. Figure 8 shows the "*Scenario 2 - Artifact size, updates available"* option selected. With this selection, the *best effort* level is the selected effort level. With the "*Artifact Details"* shopping list selection, the "*Send Artifact"* response type is not possible. The reasons for these and similar restrictions will be revealed in section 4.1.

Once the NPS options have been selected, the simulation is started.

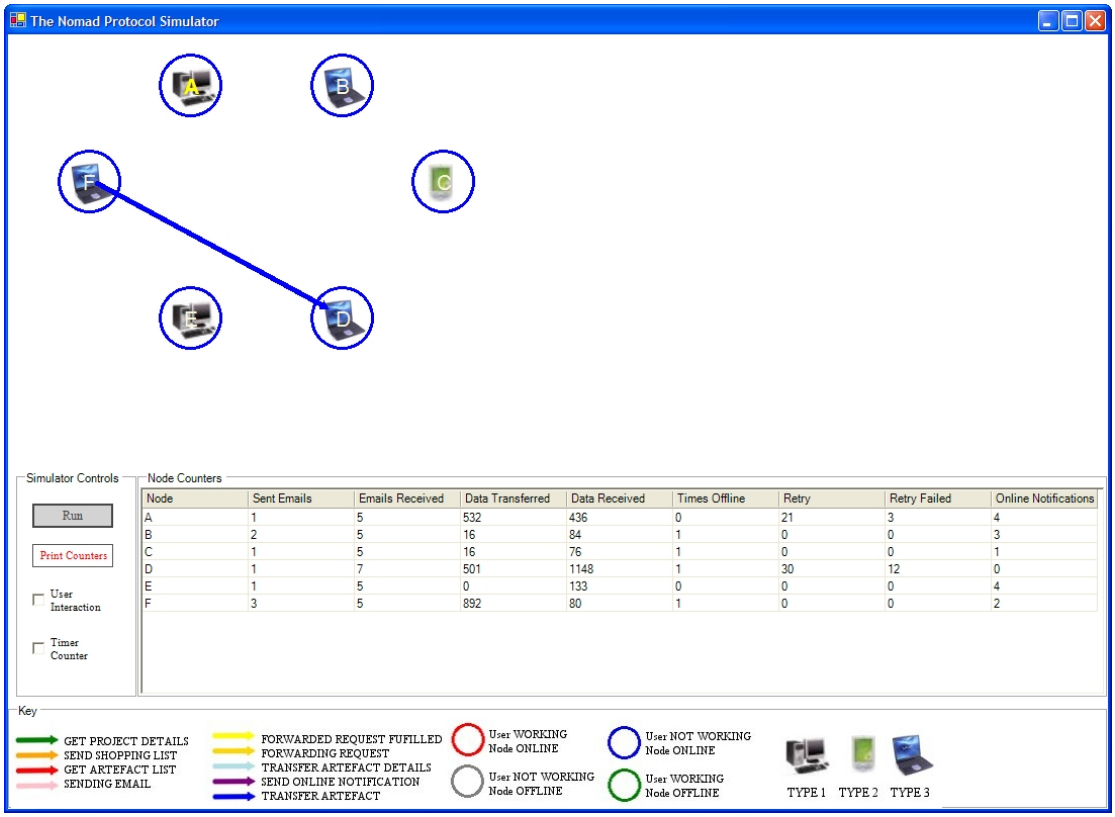


Figure 9 : Nomad Protocol Simulator form

Figure 9 illustrates the graphical NPS. The top of the canvas depicts the activity of each node in a graphical form. The NPS controls are situated on the bottom right. The user interaction allows the NPS to take a user informed decision when the option occurs. A pop up dialog box reveals the options, and based on the user's selection, the simulation continues.

When the timer counter is activated, an output file is created and the activity or *node counters* of each node is captured every *5 seconds.* This file is exported into a customized Excel Spreadsheet to reveal the graphs seen in the scenario section of this text. This will be used for analysis of the scenario.

The grid view on the bottom right gives an overview of the counters of each node. These are definable with the type of scenario. Finally, the key depicting possible items on the graphical canvas is shown at the bottom of the window.

Text output for every event that happens in the simulation during runtime is written to the console window when running with a debugger. The option exists to redirect the

output to a file if the debugging environment is not used. This file is used as a detailed analysis of the events that occurred during the simulation.

3.1.2. Definitions

There are a variety of definitions that need to be mentioned before continuing. These definitions are based on both Nomad and the NPS. Distinctions and similarities are stated were possible.

3.1.2.1. Node

A Registered Node (RN) in Nomad is a system with processing power which has been registered to be used in a Nomad project. The RN can contain multiple storage resources, made available at its own discretion, to form part of the distributed repository for the project. Each RN has a specified set of permissions to govern additions, updates and removals of other RNs and the projects structure. Each Nomad project will have its own set of RNs and together the RNs will form the entire distributed repository for that project.

The RN has the ability to run Gatherer and will process requests from the Hunter [21]. For the purposes of the simulation, a node will host the functionality of the Hunter and Gatherer process, but not make a clear definition of these two processes. The RN will be called a node or *SimNode* with respect to the NPS and through the rest of this text. The SimNode class is shown in Figure 10. Note that only relevant attributes and methods are shown.

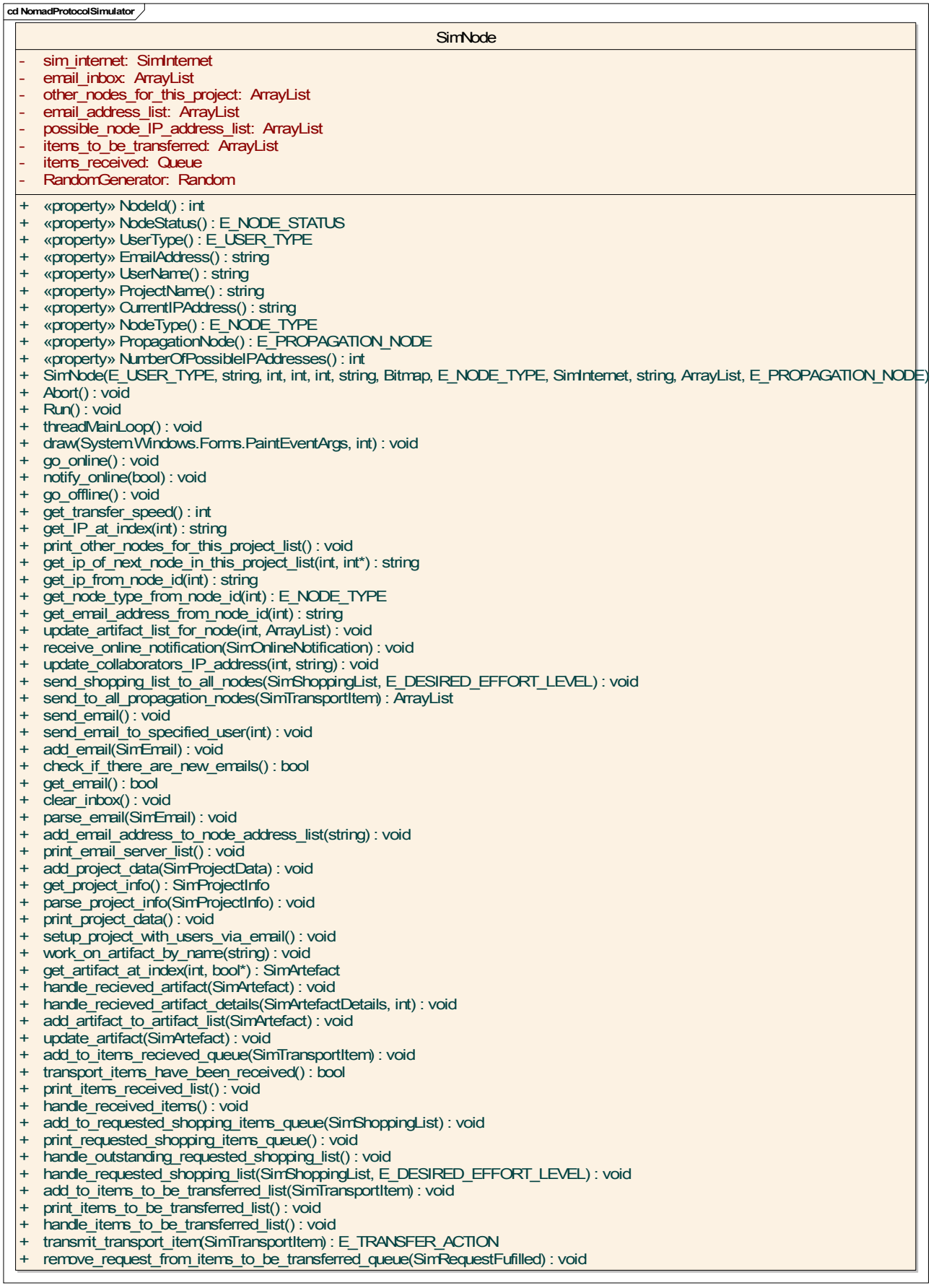


Figure 10 : SimNode class diagram

3.1.2.2. Artifacts

An artifact, in terms of Nomad, is considered to be any *"piece of information"* that relates to the project. This information could be, but not exclusive of, a paragraph of text, a picture, a diagram, audio data or documents.

An extensive amount of research relating to the description and ontology of Nomad artifacts has been done [37]. Nomad allows for artifact dependencies [21]. The work relating to artifacts in Nomad are not in the context of this text. Interested readers should reference the work discussed above.

For the purpose of the NPS, artifacts are considered to be data of any form and only exist in the *meta-data* that describe the artifact.

```
public SimArtifact(string _artifactName,
                   int    _artifactSize,
                   string _initialAuthorName,
                   int    _initialArtifactLocation)
{
        artifactName            = _artifactName;
        artifactSize            = _artifactSize;
        initialAuthorName       = _initialAuthorName;
        initialArtifactLocation = _initialArtifactLocation;
        lastAuthorUpdateName    = _initialAuthorName;
        lastArtifactLocation    = _initialArtifactLocation;
        last_update_date_time   = DateTime.Now;
        initial_date_time       = last_update_date_time;
        artifact_handled        = false;
}
```

Figure 11 : SimArtifact Constructor

Figure 11 shows the constructor for the *SimArtifact* object. When an artifact is created, we issue the name, size, author and location of the artifact. The *initial* timestamp and *modified* timestamp are the same on creation of the artifact. This timestamp is taken of current time on the workstation running the NPS. Attributes assigned within the constructor form the basis of the *meta-data* of the artifact.

When a user works on an artifact, the `lastAuthorUpdateName`, `lastArtifactLocation` and `last_update_date_time` attributes are updated to that specific user's details. The `artifactSize` is incremented by *20 Simulator Kilo Bytes(simKB)* to mimic change to the artifact.

```
public SimArtifactDetails(SimArtifact artifact)
{
   artifact_name = artifact.ArtifactName;
   artifact_size = artifact.ArtifactSize;
   artifact_last_update_time = artifact.LastUpdateDateTime;
}
```

Figure 12 : SimArtifactDetails constructor

As will be discussed in section 4.1.3, the option exists to send the details of the artifact instead of the artifact itself. Figure 12 shows the constructor of the SimArtifactDetails object.

In the NPS, an artifact is considered to be a later version if the `last_update_date_time` attribute is of a later time than the one it is being compared to. Since we are running the NPS on a single computer, the timestamp is relative to the artifacts. We note that since the simulation does not take a long period to run, the creation and update times are very close to one another as will be seen in section 4. Time differences are noted in seconds or less. In Nomad, there is a more sophisticated analysis of what is considered to be "*the latest and best artifact*" ([21],[37]).

For the purposes of the NPS, *ownership* of an artifact is not a necessity. At the time of writing, the idea was to have the project details (section 3.3.1.2 and Figure 22) include all artifact *meta-data*, and *any* user may be allowed to create an artifact. In the case of the NPS, it was not considered a major issue as it was purely an implementation concern of Nomad.

3.1.2.3. Propagation node

A propagation node is defined as any node in the community of the project that has willingly subscribed to host an item for another node. Types of items are discussed in section 3.5.1 of this text.

A propagation node has the following characteristics:

- It has typically no downtime (never offline by choice).
- It has a high storage capacity.
- It has a good internet connection.

The propagation node is useful in scenarios where nodes are in different time zones, and hence typically never online at the same time. Figure 13 illustrates this concept. After the retry count of an item is reached, *Node A* will send the item to the propagation node, *Node C* as a forwarded item. The item could be an *artifact*, or a *shopping list* request, or any item which is hosted in a *transport item*. Section 3.5.1 has a detailed description of a transport item. The propagation node allows an item to be sent on behalf of *Node A* that is about to go offline, and hosts the item until such time as the addressed node, *Node B*, comes online.

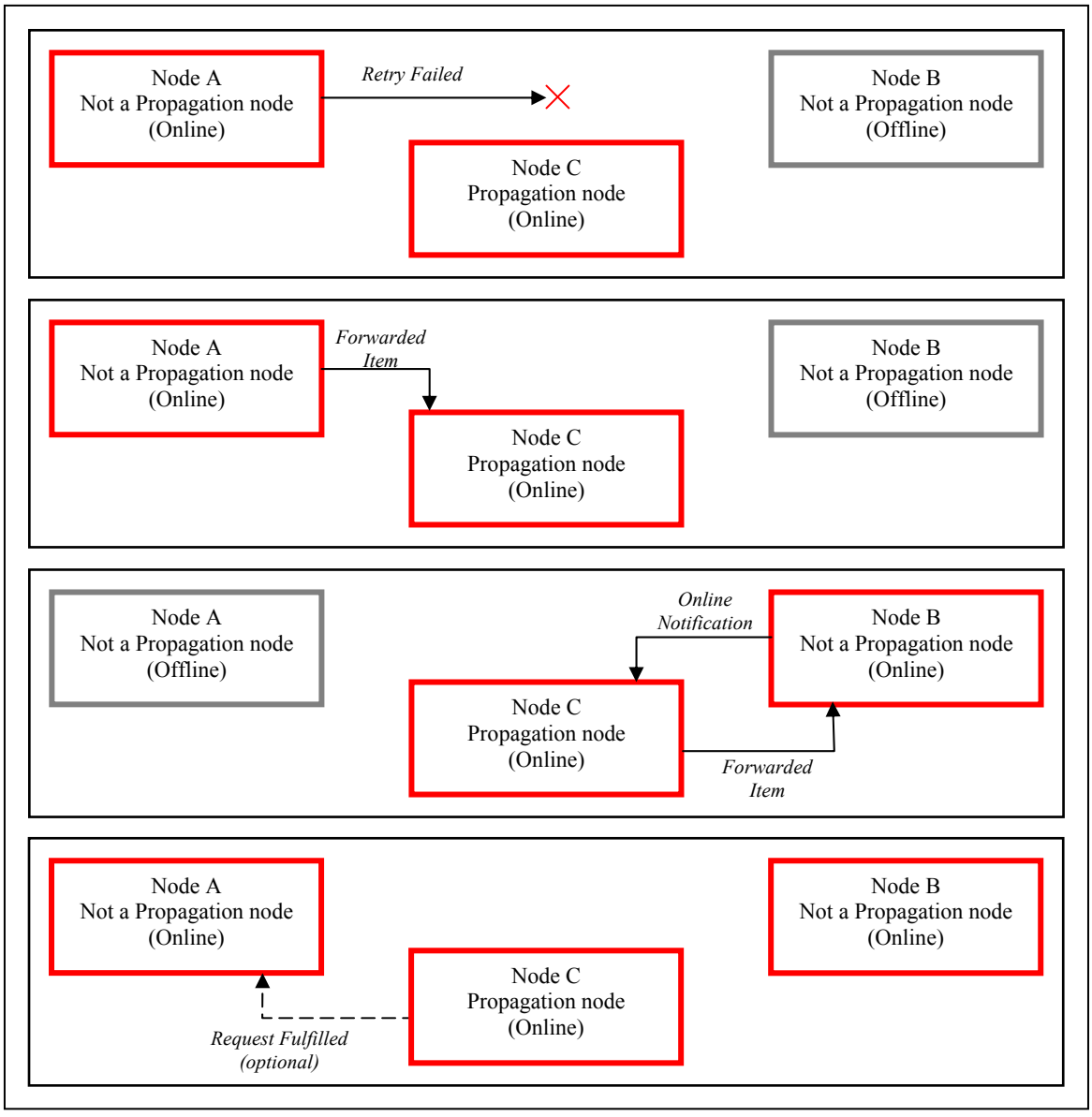


Figure 13 : Use of a propagation node

Node B will send an online notification to all nodes when it becomes available once again. The propagation node will react by forwarding the item it had been hosting for

the *Node A*. The propagation node then looks at the *meta-data* of the forwarded item, and if needed, sends a *request fulfilled* reply to the requesting node, *Node A*. This is optional tag within to the forwarded item and is illustrated in section 3.5.1.

There may be more than one propagation node in a project. But one propagation node will be sufficient for the protocol. In the NPS, a propagation node is denoted by yellow bold text, whereas all other nodes are denoted by a beige normal text.

3.2. NPS Assumptions

Since the NPS is to simulate aspects of Nomad, certain assumptions would need to be made regarding the NPS. It will not be necessary to simulate every aspect of Nomad, but core aspects would need to be implemented. The following is a list of assumptions the NPS was based on:

I. *Using one project.*

   It will not be necessary to simulate more than one project for the scope of the scenario. Every other project will be handled in the same manner as the single project illustrated in the NPS.

II. *IP addresses are defined at compile time.*

   Each node can have one or more IP address, but will be configured at compile time.

III. Based on the type of connection available and number of IP addresses available, the IP address will be randomized next time node becomes available online.

IV. A node can be in one of the following states:

   - `ONLINE_WORKING` // *Color.Red*
   - `ONLINE_NOT_WORKING` // *Color.Blue*
   - `OFFLINE_WORKING` // *Color.Green*
   - `OFFLINE_NOT_WORKING` // *Color.Gray*

   Note that this is dependant on both the user status and node status. `NOT_WORKING` and `WORKING` denotes that the user is busy updating an artifact. `OFFLINE` and `ONLINE` denotes the online status of a node.

V. *There are two types of users:*

   - `PROJECT_INITIATOR`

- `PROJECT_COLLABORATOR`

There can only be one `PROJECT_INITIATOR` but many `PROJECT_COLLABORATOR` users in a project. It is the duty of the `PROJECT_INITIATOR` to initiate first contact with other nodes via email. This aspect is further illustrated in section 3.3.1.

VI. *There are 3 types of nodes that are modeled*

- `TYPE1` (WORKSTATION)
- `TYPE2` (WIRELESS LAPTOP)
- `TYPE3` (PDA)

Each of these node types have specific attributes which will be modeled. For example, a workstation will have a permanent IP address and a good stable connection, whereas a laptop and PDA will have many IP addresses and a less stable connection. The NPS has the ability to easily integrate a new type of node if needed. The characteristics of each type are outlined in section 3.4 of this text.

VII. Since the email service plays an important part of the protocol itself, as well as the internet playing the part of the oracle (one who knows all), these two systems have been modeled in the NPS. The *SimInternet* object plays the part of knowing the state of all nodes in the simulator as well as creating connections and transferring any form of "data" between nodes. Data is in the form of emails, shopping lists, artifacts, project details, etc. These data items have been further encapsulated into a *SimTransportItem*.

VIII. *Simulator Kilo Bytes (simKB)* .

The NPS uses Simulator Kilo Bytes (simKB) to signify the data unit used in simulator. Values of interest are shown in this thesis where appropriate..

We use enumerations as defined in Figure 14 to illustrate concepts relevant to the NPS. Only relevant enumerations are listed here. These enumerations are explained within relevant sections of this text.

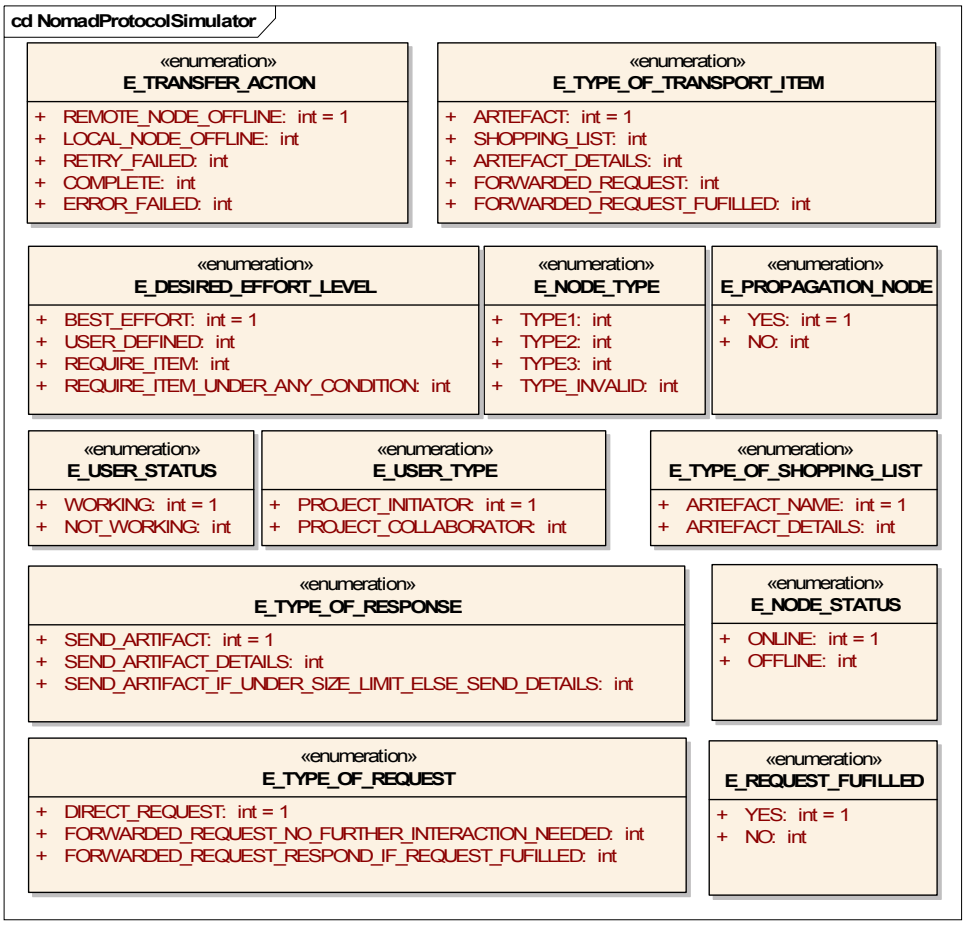


Figure 14 : Enumerations used in the NPS

3.3. Nomad Protocol Properties

This section relates the Nomad protocol to how the NPS is implemented. Where necessary, this section begins by explaining how the Nomad protocol is expected to be implemented, and how it has been implemented in the NPS. Certain properties are only valid for the NPS, but mention is made to the relation of Nomad.

3.3.1. Setting up the community

For each project, communities of users are set up. This section illustrates the technologies and technique of setting up a project.

3.3.1.1. According to the Nomad Protocol

The Nomad protocol relies on the email infrastructure to set up the community. A node may belong to one or multiple projects. There exists one *Project Initiator* and many *Project Collaborators* within a project. The onus is on the Project Initiator node

to send the initial email to all Project Collaborators. Figure 15 illustrates the information embedded within the email.

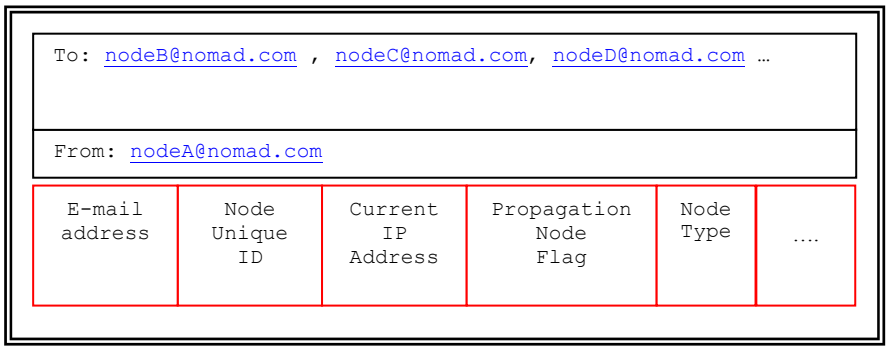


Figure 15 : Senders information embedded within email

Once this is done, the Project Initiator node reacts like any other Project Collaborator node. Once the Project Collaborator has received the setup email, Nomad parses the email, which contains details of the Project Initiator. The Project Collaborator node then contacts the Project Initiator to get the project details. The project details contain the email addresses of all other nodes that belong to the project, and other project related details, such as the list of artifacts within the project.

The Project Collaborator node sends an email to all nodes within the project. When a Project Collaborator node receives an email, it parses the email. The email contains the details of the other nodes name, IP address and other relevant information. Once all nodes have sent emails to every other node in the list, the project is setup. Nodes may enter and leave the community at any time. Bishop [21] further discusses this process.

3.3.1.2. Implementation of the Nomad Protocol Simulator

Figure 16 illustrates how the NPS sets the nodes at startup.

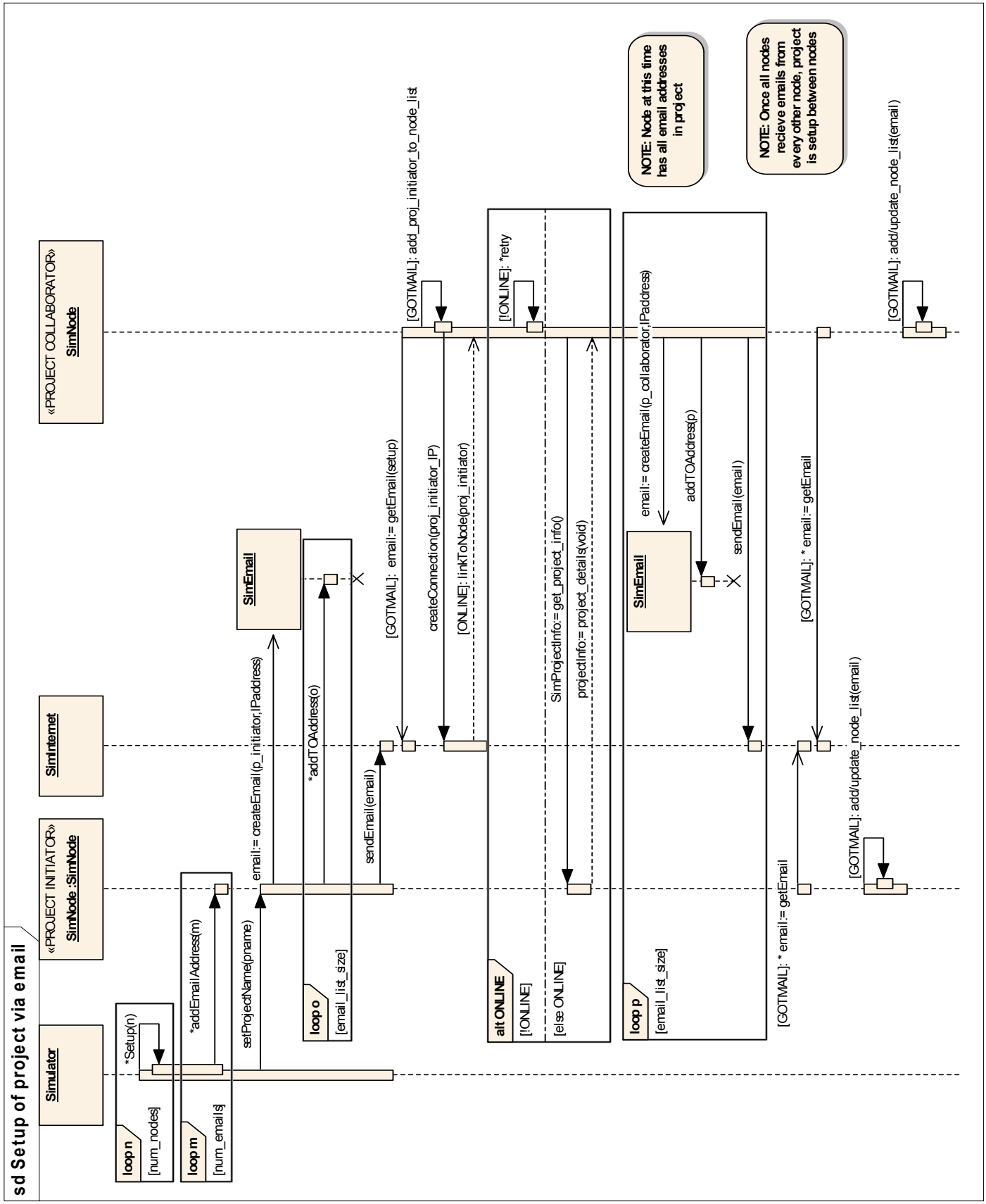


Figure 16 : Simulating the setup of a Nomad project via email

As can be seen, the NPS follows the setup of Nomad closely. It should be noted that within the NPS, only *one* project is set up. The same functionality would apply to other projects that are setup. Adding more than one project would not be advantageous at this point to the simulator.

The other aspect of importance would be the handling of email between nodes. Since emails are used for a variety of reasons within Nomad, it would need to be simulated within the NPS. Emails are sent at the initial project set up. They are also used to keep nodes in sync with respect to IP addresses when they go offline during the same period of time and go online with different IP addresses. This is further discussed in the scenarios of section 4. Figure 17 illustrates the class diagrams for the classes used to simulate email within the NPS.

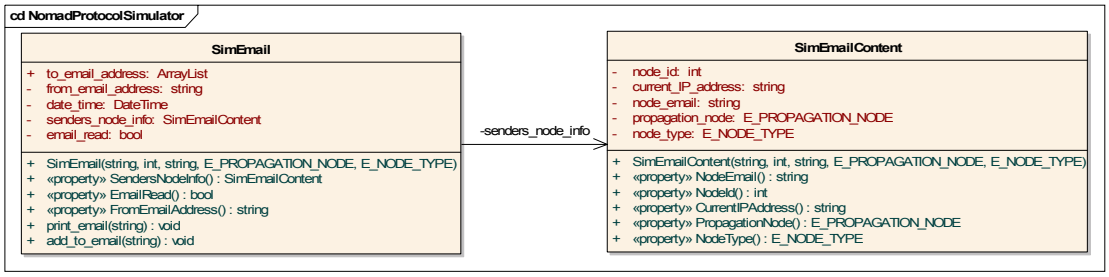


Figure 17 : SimEmail and SimEmailContent class diagrams

Figure 18 show the constructors for the SimEmail and SimEmailContent objects of the NPS. A single email can be sent to multiple addresses. The SimEmailContent object acts as a wrapper for the particulars of the SimEmail object.

```
/* SimEmailContent constructor*/
public SimEmailContent(string _node_email,
                       int _node_id,
                       string _current_IP_address,
                       E_PROPAGATION_NODE _propagation_node,
                       E_NODE_TYPE _node_type)
{
       node_email         = _node_email;
       node_id            = _node_id;
       current_IP_address = _current_IP_address;
       propagation_node   = _propagation_node;
       node_type          = _node_type;
}

/* SimEmail constructor*/
public SimEmail(string _from_email_add,
                int _node_id,
                string _current_IP_address,
                E_PROPAGATION_NODE _propagation_node,
                E_NODE_TYPE _node_type)
{
       /* create the array of addresses this email will be sent to*/
       to_email_address = new ArrayList();
       to_email_address = ArrayList.Synchronized(to_email_address);

       from_email_address = _from_email_add; /* info of sender */
       date_time = DateTime.Now;             /* timestamp      */
       email_read = false;                   /* new email      */

       /*create the SimEmailContent for this email */
       senders_node_info = new SimEmailContent(_from_email_add,
                                               _node_id,
                                               _current_IP_address,
                                               _propagation_node,
                                               _node_type);
}
```

Figure 18 : SimEmail and SimEmailContent constructors

Any node has the ability to create, send and receive an email. The email will be sent to all nodes within the project. Within the NPS, a *SimNode* creates an instance of *SimEmail* and sent through the *SimInternet* object to the inbox of the addressee. Figure 19 illustrates the simulation of email between *SimNodes*.

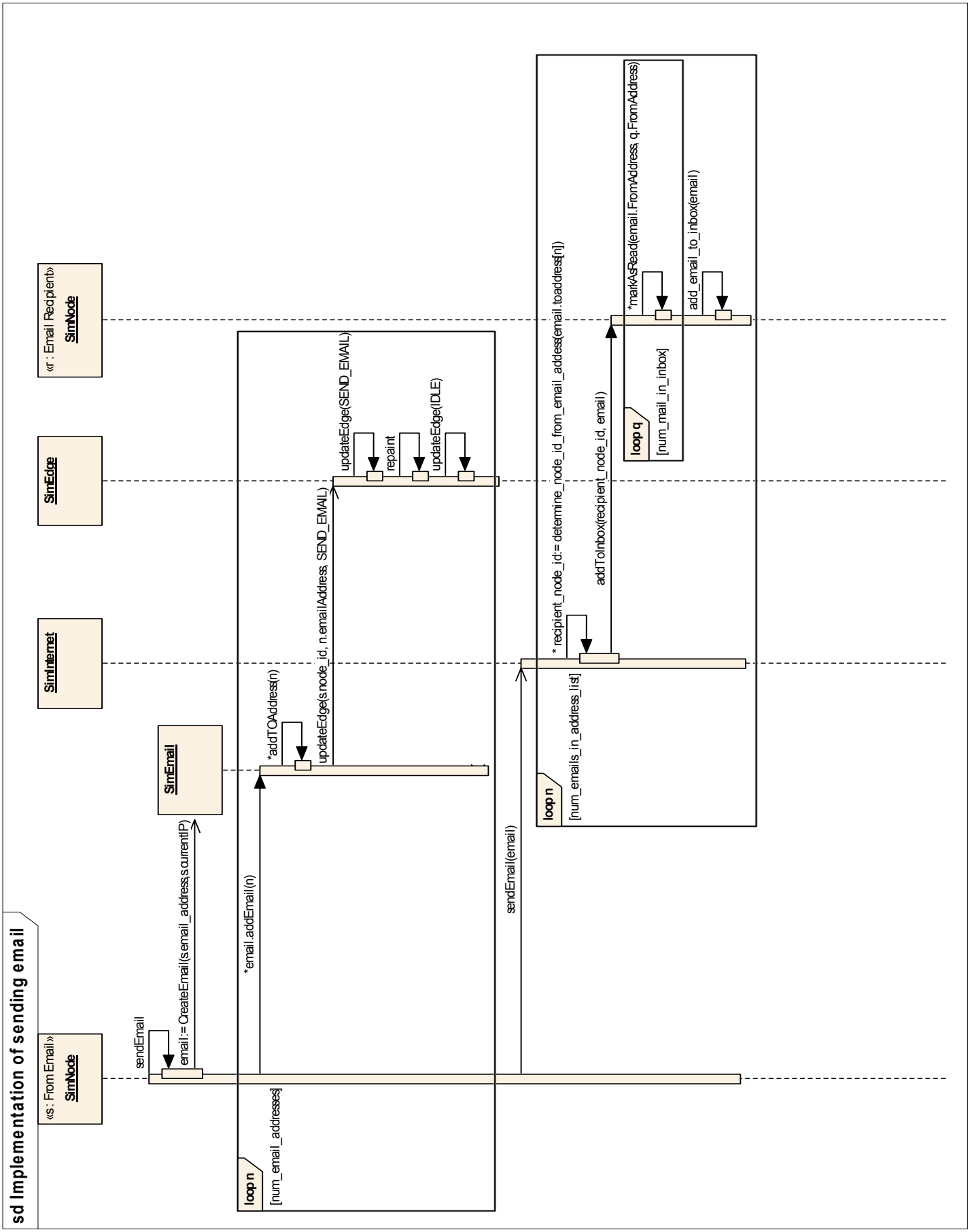


Figure 19 : Simulating sending of email between nodes

```
public void getEmail()
{
   if (email_inbox.count > 0)
   {
      foreach (SimEmail email in email_inbox)
      {
         /* only read most recent emails */
         if (email.read == false)
         {
            if(email.senders_node_info.node_id in
               other_nodes_for_this_project)
            {
                updateIP(email.senders_node_info.node_id,
                         email.senders_node_info.current_IP_address);
            }
            else
            {
               /* New User*/
               this.other_nodes_for_this_project.add(
                              email.senders_node_info.node_id,
                              email.senders_node_info.current_IP_address);
               email.email_read = true;
            }//end else
         }//end if
      }//end foreach
      email_inbox.clear();
  }//endif
}//end getEmail()
```

Figure 20 : Pseudo code for the NPS getEmail function

Figure 20 illustrates pseudo code for simulation of receiving email in the NPS. Once the set up email has been received by the collaborator node, the collaborator node then contacts the Project Initiator to get the project details. We define the detailed project information in the NPS as depicted in Figure 21.

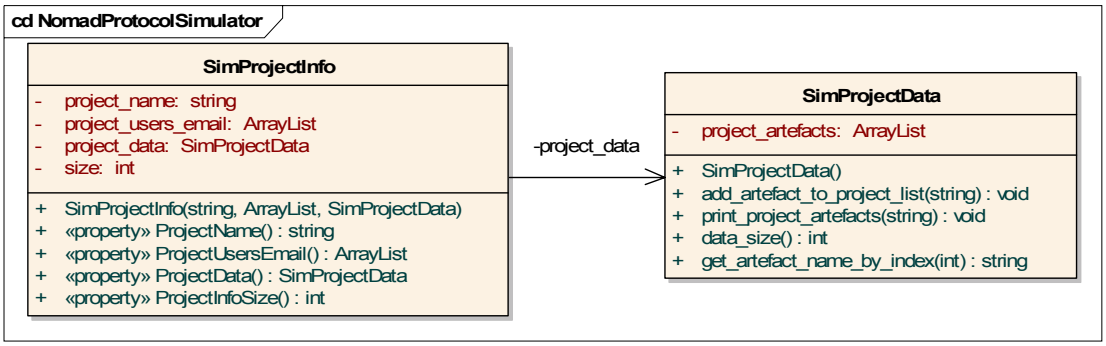


Figure 21 : Project detail classes as in NPS

The project details that are distributed within in the NPS are shown in Figure 22.

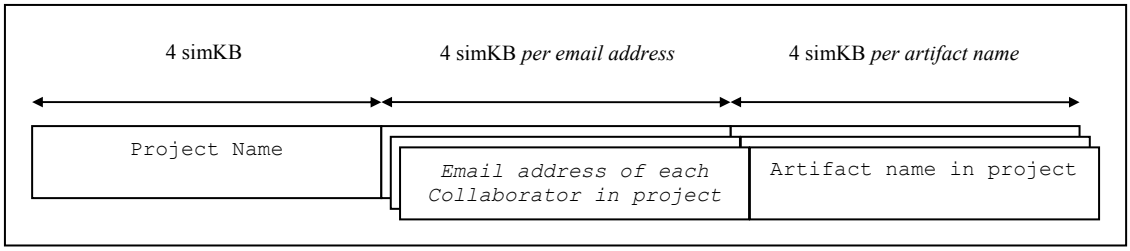


Figure 22 : Project details

The project has one name, a number or collaborators and a number of artifacts. The artifacts are described by unique names. For the purposes of the scenarios in the NPS for this text, all collaborator email addresses and artifact names are sent initially and are not updated thereafter. These details can however be updated at any time. It should be noted that each scenario in this text assumes the same startup procedure, in which the project details are sent at startup.

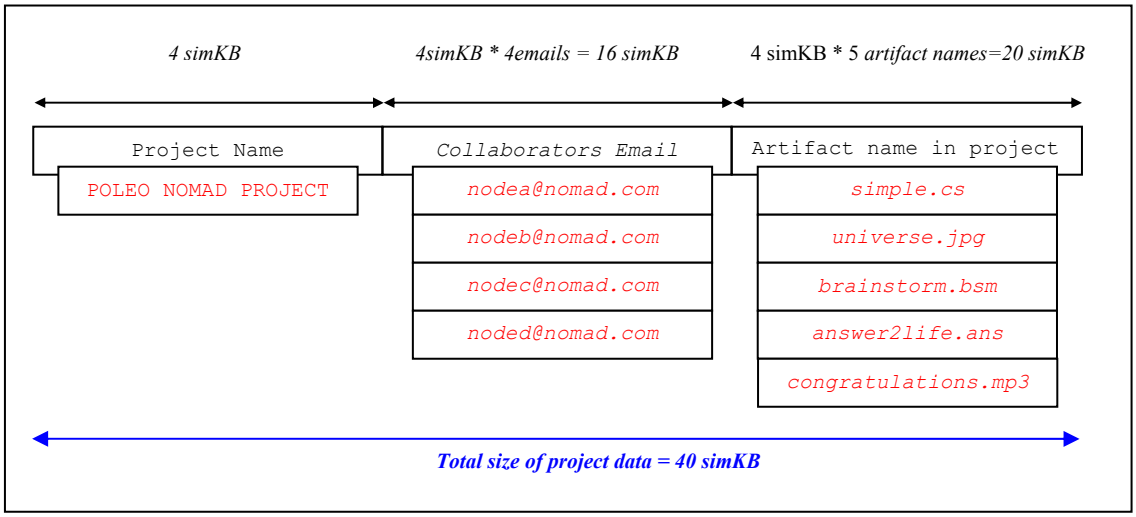


Figure 23 : Typical project data for the NPS

Figure 23 illustrates a typical setup for the project data. The artifact names and emails addresses are arbitrary and do not relate to any real life project and data.

3.3.2. Simulating Node Connectivity

An important aspect of the simulator is the ability to simulate the nodes online status. This represents the online availability of the node and simulates the node connecting to the community via an internet connection point.

3.3.2.1. Node available online

When a node becomes available online, and before it connects to the community, it has to simulate acquiring of a new IP address and updating any new information about other nodes which has become available via email. Each *type* of node follows a different setup procedure to acquire the new IP address from the list of available IP's. Node types are discussed in section 3.4. The lists of IP addresses are *hard coded* at compile time and any number of IP addressees may be assigned. Once a node has been assigned with a new IP address, it checks for any new email. Figure 24 illustrates the functionality implemented in the NPS when a node goes online.

```
public void goOnline()
{
    bool thisNodesIPChanged = false;
    string previous_IP_address = this.CurrentIPAddress;

    /* -------- workstation ------------ */
    if(this.NodeType == E_NODE_TYPE.TYPE1)
    {
        thisNodesIPChanged = false;
    }
    /* ----------- laptop -------------- */
    else if (this.NodeType == E_NODE_TYPE.TYPE2)
    {
        rand = this.Random(10);
        if (rand > 5) /*50% chance of changing IP address*/
        {
            this.CurrentIPAddress = Random(possible_node_IP_address);
            thisNodesIPChanged = true;
        }
        else
        {
            thisNodesIPChanged = false;
        }//end else
    /* ------------ PDA --------------- */
    else if (this.NodeType == E_NODE_TYPE.TYPE3)
    {
        /* randomize to a new IP address */
        while (this.CurrentIPAddress == previous_IP_address)
        {
             this.CurrentIPAddress = Random(possible_node_IP_address);
            thisNodesIPChanged = true;
        }//end while
    }//end else

    /* Check if new email and update if necessary*/
    if (email_inbox.count > 0)
    {
        getEmail();
    }//end if

    /* Notify other collaborators that node is online*/
    notifyOnline(thisNodesIPChanged);

}//end goOnline()
```

Figure 24 : Pseudo code for the NPS goOnline function

3.3.2.2. Notify online

Once the node is online, it will contact the community to let them know that it is available. The node tries to send an *online notification* to each collaborator in the community. If it is unable to connect to the collaborator, since the collaborator might be *offline*, the simulator checks if the *user interaction* is enabled.

If *user interaction* is enabled, the system asks the user if an email with the new IP address should be sent to the collaborator that is not available. Depending on the choice of the user, the system either sends or does not send an email. The choice has been added since a user might have a `TYPE3` node (PDA) and will not be online for a long period of time. Therefore, it will be a futile task to send an email if it returns to being offline soon.

If the *user interaction* is disabled, the system makes a decision dependant on the *type* of node it is. If it is a node of `TYPE1` or `TYPE2`, then the system will send the email with the updated information. If it is a node of `TYPE3`, the system does not send the email with the updated information.

```
public void notifyOnline(bool thisNodesIPChanged)
{
    bool sent = false;
    bool remote_node_online = false;

    SimOnlineNotification notify = new SimOnlineNotification(this.NodeId,
                                                    this.CurrentIPAddress);

    /* attempt to send notification to each remote collaborator node*/
    foreach (SimNode remote_node in other_nodes_for_this_project)
    {
        remote_node_online = connect_to_remote_node(remote_node);
        if (remote_node_online == true )
        {
            send_online_notification(notify,remote_node.NodeId)
        }
        else /* remote node offline */
        {
            if (thisNodesIPChanged == true)
            {
                if (sim_internet.check_user_interaction_status() == ENABLED)
                {
                    /* dialog box requiring YES or NO response*/
                    choice =  promptUser("Send email to offline user with
↳your new IP Address?");
                    if (choice == Yes)
                    {
                        /* send an email to the offline remote node with*/
                        /*    this nodes new IP address */
                        send_email_to_specified_user(remote_node.NodeId);
```

```
                    }
                }
                else //automated
                {
                    /* If this is a workstation or laptop */
                    if ((this.NodeType == E_NODE_TYPE.TYPE1)
                       || (this.NodeType == E_NODE_TYPE.TYPE2))
                    {
                        /* send an email to the offline remote node with */
                        /* this nodes new IP address                     */
                        /* so when remote offline node comes online,     */
                        /* will check IP and update their list           */
                        send_email_to_specified_user(remote_node.NodeId);
                    }/* else if this is a PDA*/
                    else if (this.NodeType == E_NODE_TYPE.TYPE3)
                    {
                      /* then don't send email, since a node of this type */
                      /*will probably be online  for short period of time */
                    }//end else
                }//end else
            }//end if /* thisNodesIPChanged */
        }//end else /* remote node offline */
    }//end foreach
}//end notifyOnline()
```

Figure 25 : Pseudo code for the NPS notifyOnline function

Figure 25 illustrates the functionality implemented for sending *online notifications.* An online notification is simply the unique node id and the current IP address of the node.

```
/*create an online notification */
SimOnlineNotification notification = new SimOnlineNotification
                                                (this.NodeId,
                                                 this.CurrentIPAddress);
```

Figure 26 : Creating an SimOnlineNotification

Figure 26 shows a simple creation of a *SimOnlineNotification*. Since the online notification is meant to be sent many times during a real life situation, and therefore we aim to keep it as simple and small as possible. In the NPS, an online notification counts as 4 simKB.

3.3.2.3. Node offline

A node simply changes its status to `OFFLINE` when it goes offline. The reason this choice is so simplistic, is since it relates to real life as closely as possible. If a node crashes or has a power loss, Nomad should have the ability to handle this situation. This can happen regardless of the type of node.

3.4. Node Types

After an intensive research into types of devices and protocols used to connect to the internet, we categorize the types of devices into 3 classifications. For the purposes of this text, the characteristics of each type described below are assumed to be true. Due to the ever increasing processing power, mobile devices become more powerful. The descriptions given below may not be accurate, but the classification is valid and made for simplicity. The NPS has the ability to add a new type of node if necessary.

3.4.1. TYPE 1 – Workstation

The workstation forms the motivation of this type. This type of device is generally online for long periods of time with very short, if at all down times. The workstation is generally powered on for most of its lifetime.

The workstation is normally connected to a Local Area Network (LAN). The LAN has the characteristics of having an *excellent* internet connection. The connection is generally fast, stable and always available. The assigned IP address of the workstation is normally unchanged, even if the system was temporarily offline.

There is an abundance of available space, processing power and memory on the workstation. In the NPS, the simulated transfer speed for a node of TYPE1 is 30 *simKB/sec*.

3.4.2. TYPE 2 – Mobile Laptop

The mobile laptop forms the motivation of this type. This type of device is generally online for long periods of time with many short down times. This could relate to a laptop being moved from work to home or vice versa.

The laptop is typically connected to a LAN at work and a dialup connection at home. The dialup connection could be through a normal phone line. The dialup connection has the characteristics of having an *average* internet connection. The connection is average, stable and always available on demand. The assigned IP address of the laptop is normally changed when the system moves between places. It therefore has at least two IP addresses.

There is an *average* amount of available space, processing power and memory on the laptop. In the NPS, the simulated transfer speed for a node of TYPE2 is 20 *simKB/sec*.

3.4.3. TYPE 3 – Mobile PDA

The mobile Pocket Digital Assistant (PDA) forms the motivation of this type. This type of device is generally online for short periods of time with large periods of down time.

The PDA or similar mobile device typically connects to the internet via a Bluetooth enabled cellular phone or the device might have its own *wireless* connectivity. Connection in made by means of General Packet Radio Switching (GPRS) or 3G (3rd Generation Networks). The GRPS and 3G connections have the characteristics of having a generally medium to slow internet connection. Most coffee shops offer wireless internet connections. The connection is generally expensive and therefore only lasts short periods of time. The connection is poor, unstable and available only where there is coverage. The assigned IP address of the PDA is never the same as it roams between different connections.

There is a very *small* amount of available space, processing power and memory on the PDA. In the NPS, the simulated transfer speed for a node of TYPE3 is 10 *simKB/sec*.

3.5. Management queues

The NPS simulates two management queues. An `Items_to_be_transferred` queue and an `Items_received` queue. The `Items_to_be_transferred` queue manages the items that are transferred from a specific node to other nodes. The `Items_received` queue manages the items that are received from other nodes to a specific node. We define a *SimTransportItem* which acts as a wrapper for any type of item that is sent between nodes. We give an overview of how these items fit into the context of the NPS.

3.5.1. Transport Item

In the initial design of the NPS, for the sake of simplicity we opted to transfer individual items via the *SimInternet* object. The problem arose when the number of

items increased. With every new item defined, a new method defining the transferring of the object in the *SimInternet* object had to be written.

We then introduced a *SimTransportItem* which served as a wrapper for any item that was required to be transferred between nodes. We therefore had a single method in the *SimInternet* object which meant a single entry point for any item to be transferred between nodes. Figure 27 shows the constructor of the *SimTransportItem*.

```
public SimTransportItem(int _from_node,
                        int _to_node,
                        E_TYPE_OF_TRANSPORT_ITEM _type,
                        E_DESIRED_EFFORT_LEVEL   _effort_level)
{
       /* populate local variables */
       from_node    = _from_node;
       to_node      = _to_node;
       type_of_item = _type;
       effort_level = _effort_level;

       /* default charateristics of the Transport Item*/
       number_of_retries = 0;
       item_handled = false;

       /** by default - the type of request is a DIRECT_REQUEST
       *- only changed when request needs to be sent to
       *  propagation node*/
       type_of_request     = E_TYPE_OF_REQUEST.DIRECT_REQUEST;

       /** default for forwarded requests - updated on change of
       * type_of_request */
       forwarded_from_node     = _from_node;
       forward_to_node         = _to_node;
       forward_request_id      = -1; /*default*/
       type_of_forward_request = E_TYPE_OF_REQUEST.DIRECT_REQUEST;
}
```

Figure 27 : SimTransportItem Constructor

Although it simplified the scalability of introducing new items, we observed an increase in overhead for the *SimInternet* object since it is a shared object. The overhead had an unnoticeable effect on performance. However, the shared object allowed only a single item to be transferred between nodes at any one point in time. This was acceptable since the NPS is based on events.

Figure 28 illustrates the *SimTransportItem*. The meta-data consists of origin and destination of the item, type of item, effort level of delivering the item, a counter for the number of times this item was attempted to be sent but failed, a handle flag and the type of request.

If the item is a *forwarded item* to a propagation node, additional meta-data is populated describing information pertaining to the forwarded item.

The size of the item is dependant on the item being transferred. The total size is made up from the item and an additional *20simKB* for overhead of the class and wrapper.

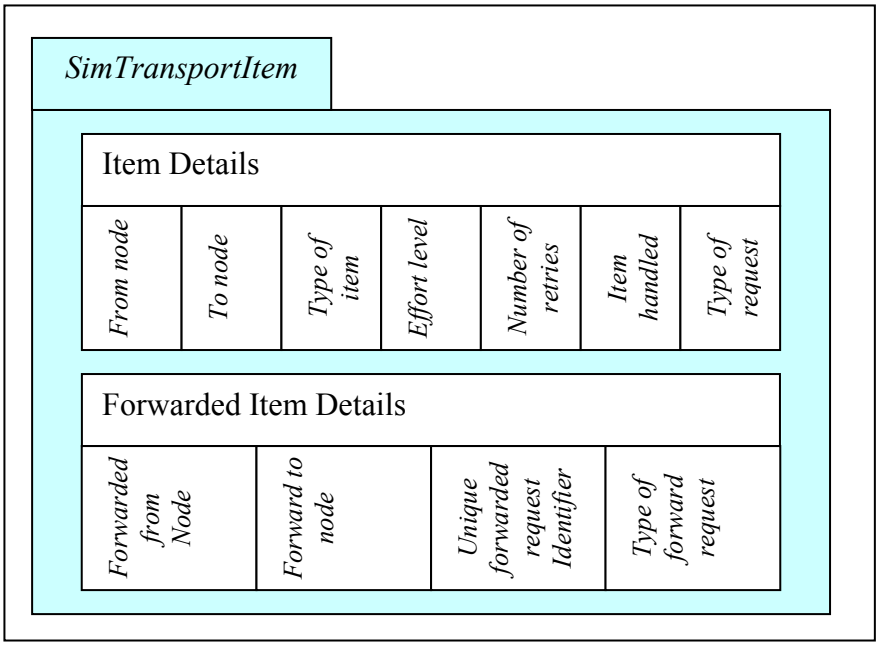


Figure 28 : SimTransportItem

A single item is wrapped within a *SimTransportItem*. The item could be one of the following:

- `ARTIFACT`,
- `SHOPPING_LIST`,
- `ARTIFACT_DETAILS`,
- `FORWARDED_REQUEST`,
- `FORWARDED_REQUEST_FUFILLED`

A `FORWARDED_REQUEST` is an item that has been sent to a propagation node and therefore has an embedded *SimTransportItem* which will be forwarded to the required node. A `FORWARDED_REQUEST_FUFILLED` item is sent on behalf of the propagation node to the requesting node (*Forwarded from Node* ID), denoting the success or failure of forwarding the item to the desired node (*Forward to node* ID). Figure 29

illustrates the relation of the *SimTransportItem* to other classes and enumerations in the NPS.

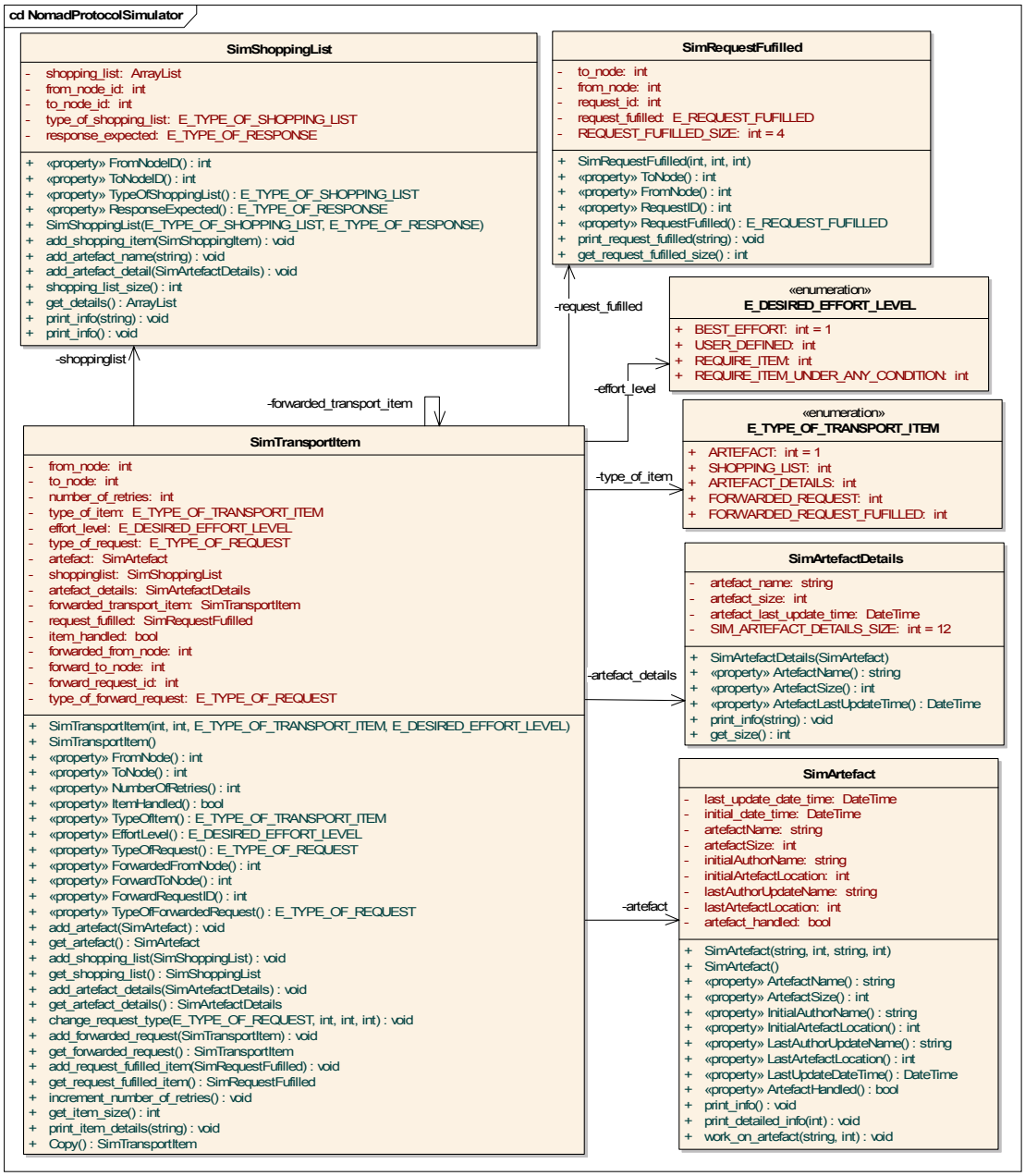


Figure 29 : SimTransportItem and relation to NPS classes

3.5.2. Transfer and Receive queues

Once a *SimTransportItem* is created, the item is placed on the *Items to be transferred* queue. The queue is responsible for transferring items to the required nodes via the *SimInternet* object. It maintains the retry count of each item and increments the "*number of retries"* counter on each failure of transferring an item.

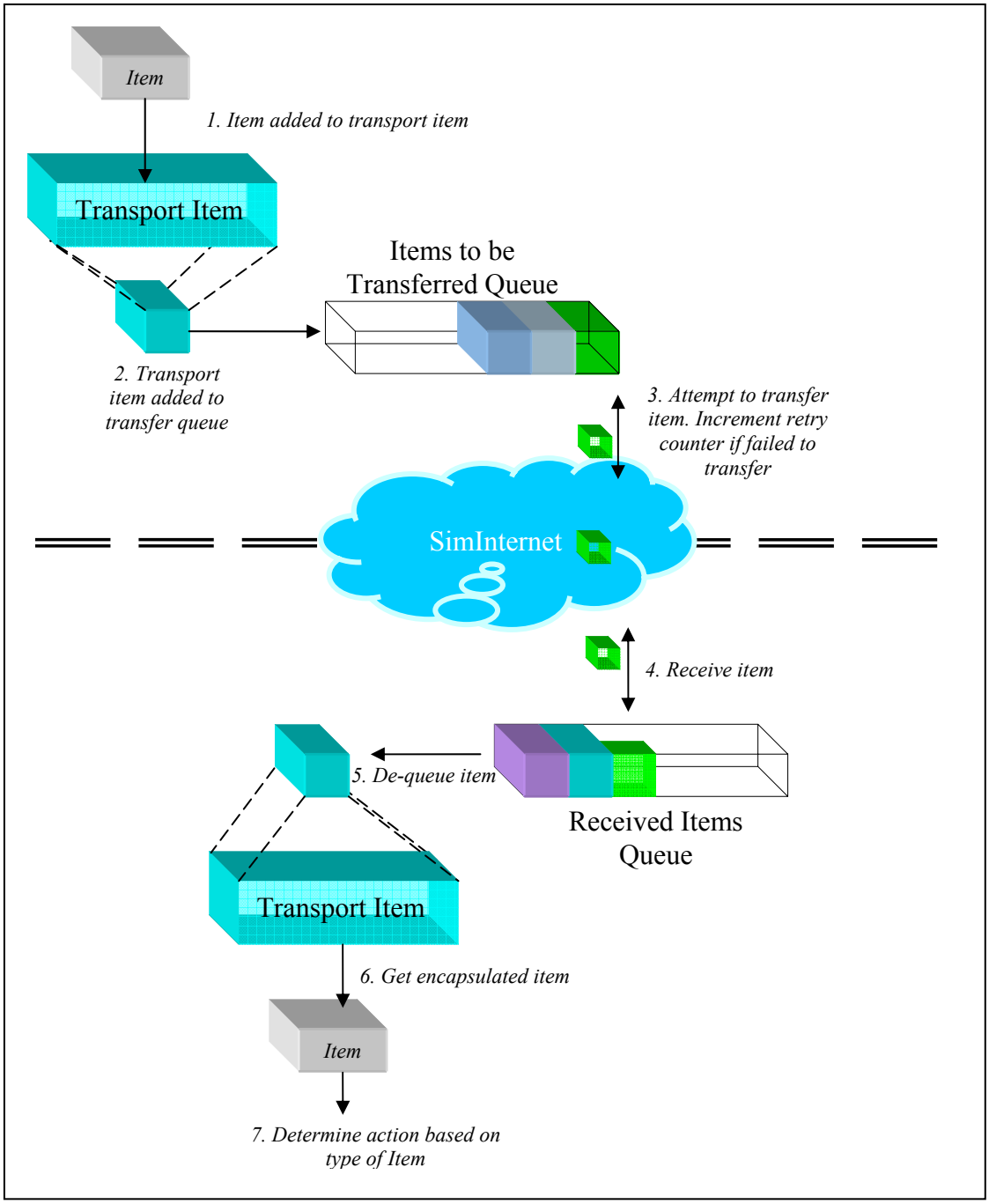


Figure 30 : Management Queues

On a successful transfer, the item is added to the receiving nodes *Items Received* queue. The embedded item is then identified and handled accordingly. Figure 30 illustrates a high level concept of the management queues. Although finer grain details such as error checking is not shown, they have been implemented in the NPS.

4. Simulation Scenarios

Every scenario has the first node, generally *Node A,* defined as the *Project Initiator*. All other nodes are defined as *Project Collaborators*. Each scenario creates the community as defined in section 3.3.1.2. Hence the figures related to data transfer and the number of emails includes the overhead caused by setting up of the project.

4.1. Influence of Artifact size

The initial thought process behind sending of artifacts is depicted in Figure 31. What is noted is that the artifact name is processed and the artifact is sent across the network. Since Nomad is to be used initially with small artifacts, such as lines of code or small paragraphs, the chances are that these artifacts are fairly small and should be transferred immediately across the network.

If the artifacts were small, they would not have a major effect on other online nodes. But what if the size of these artifacts became larger? Since there is no version checking on which artifact to send, this might cause a heavy overhead on the other collaborator nodes. The worst case scenario is that a node gets an older artifact from another collaborator. This wastes time and effort of both nodes.

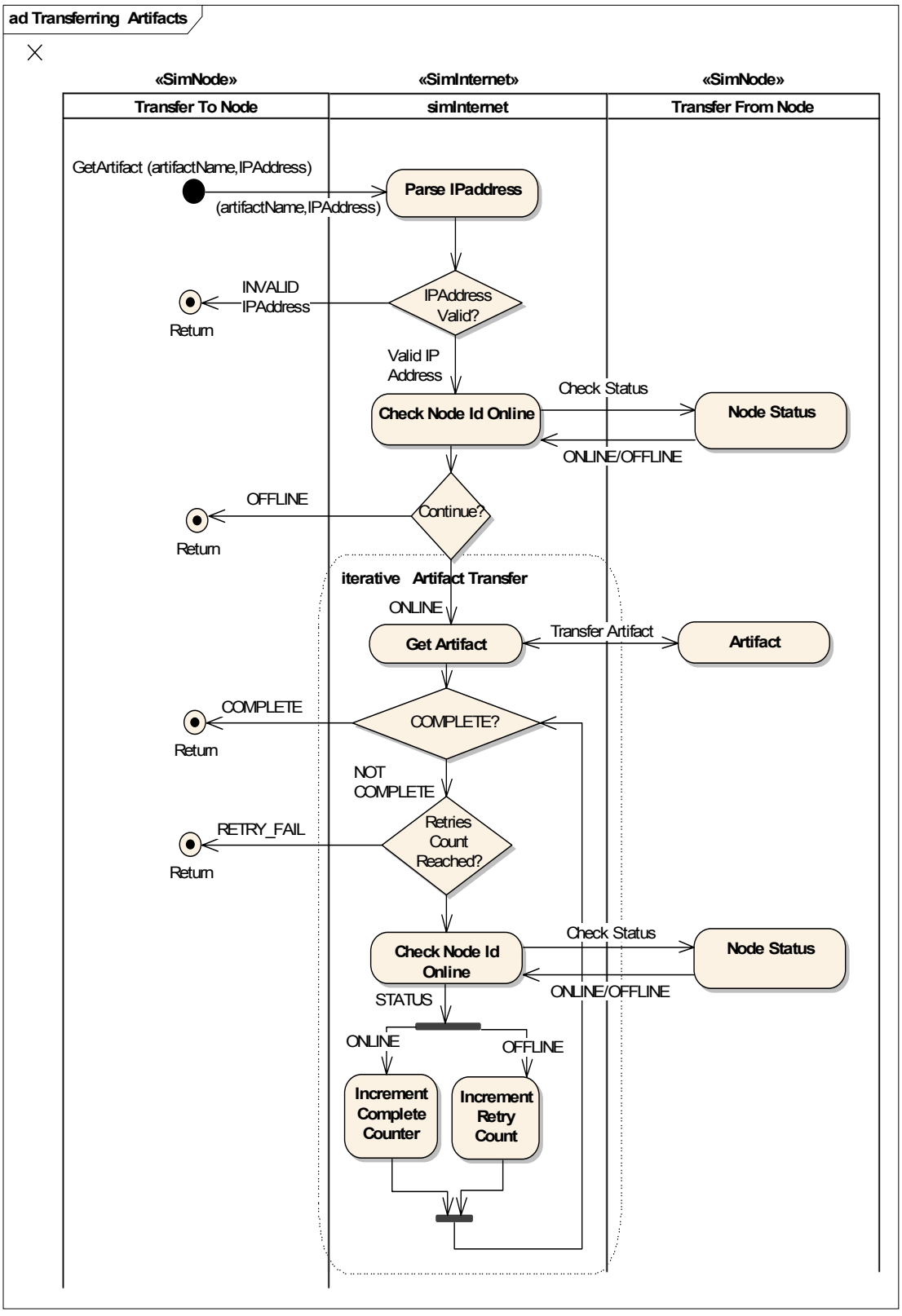


Figure 31 : Initial transfer of artifacts

4.1.1. Scenario setup and description

This scenario is made up of the following nodes:

Node name	Node Type
A	TYPE1
B	TYPE1
C	TYPE1
D	TYPE1

Table 2 : Artifact Size Scenario Setup

A visual description from the NPS is shown in Figure 32.

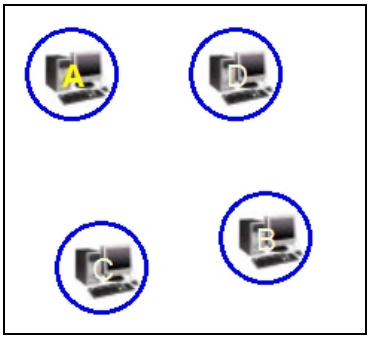


Figure 32 : NPS Artifact Size Scenario

This simulation is used to show the effect of artifact size and transfers over the protocol. All nodes are workstations. There are no disconnections. The main idea is to show the amount of data transferred for the *artifacts only*, *artifact details* and *send artifact if under size limit else send artifact details* options in the NPS.

When the simulation starts, the project is setup amongst the nodes. *Node D* then creates two artifacts, namely *"Simple.cs"* and *"Universe.jpg"*. *"Simple.cs"* is a small artifact of size *100 simKB* and *"Universe.jpg"* is a large artifact of size *1000 simKB*.

Node A proceeds to create the same artifact *"Simple.cs"* and works on the artifact, hence having the most updated version of *"Simple.cs"*. *Node B* proceeds to create the artifact *"Universe.jpg"* and works on the artifact, hence having the most updated version of *"Universe.jpg"*. We translate *the most updated version* of the artifact, as an

artifact that has the latest modification timestamp attached to the artifact. For further details on artifacts, see section 3.1.2.2 of this text.

If the *no update* scenario is run, *Node D* works on *"Simple.cs"* and *"Universe.jpg"*. It therefore has the most updated versions of these two artifacts and therefore does not need an update from any other collaborator.

Node D creates a shopping list for *"Simple.cs"* and *"Universe.jpg"* and sends the shopping list to all collaborators. For the simulations that follow, the type of shopping list and type of response differ.

There are two types of shopping lists, a*rtifact name* and *artifact details*. For the latter case, for the purposes of the simulation, the requesting node has to have the artifact existent. The reason is that when the details of the artifact that exists on the collaborator node are sent back to the requesting node, the requestor has to have something to compare with.

There are three types of responses:

- Send the artifact that exists on the collaborator node;
- Send details of the artifact that exists on the collaborator node; or
- Send the artifact if it is under a certain size, else send details of the artifact that exists on the collaborator node.

These responses are further discussed in the sections that follow.

4.1.2. Requesting and transferring artifacts only

In this situation, the requestor broadcasts its request to all available collaborator nodes. The request simply contains the artifact name. If the node has the requested artifact, it sends the artifact, regardless of the age of the artifact. Figure 33 illustrates this concept.

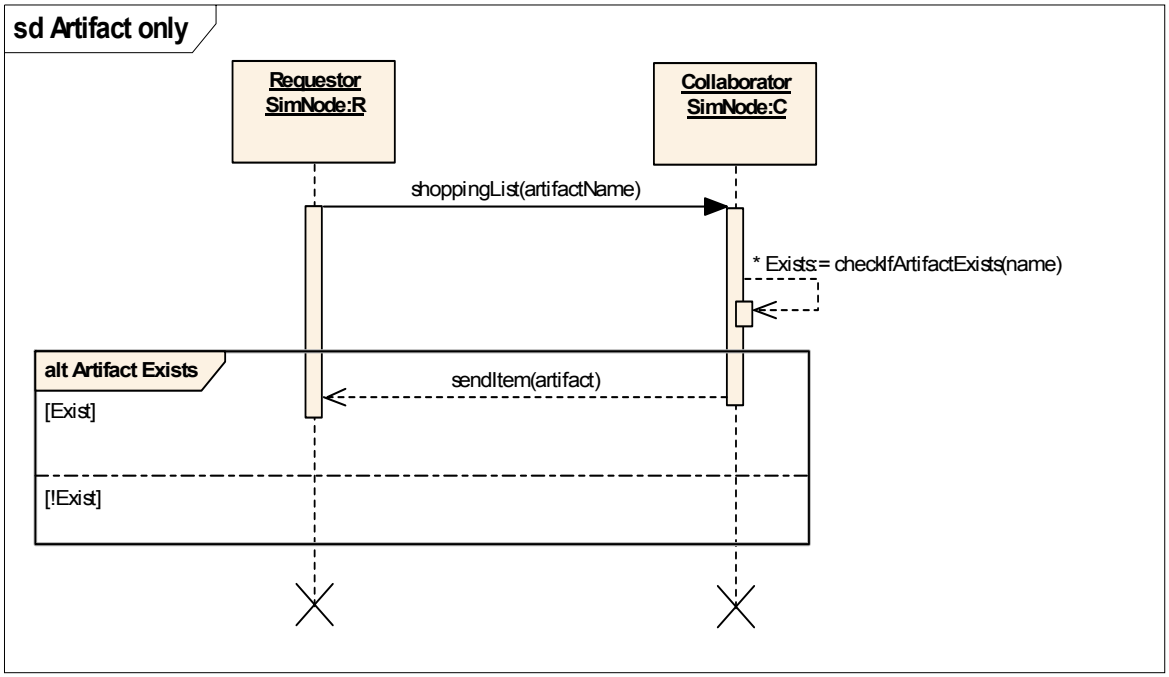


Figure 33 : Send artifact

If updates exist on other nodes, then the total data transferred from other nodes are as in Figure 34. *Node A* transferred the updated version of *"Simple.cs"*. *Node B* transferred the updated version of *"Universe.jpg"*.

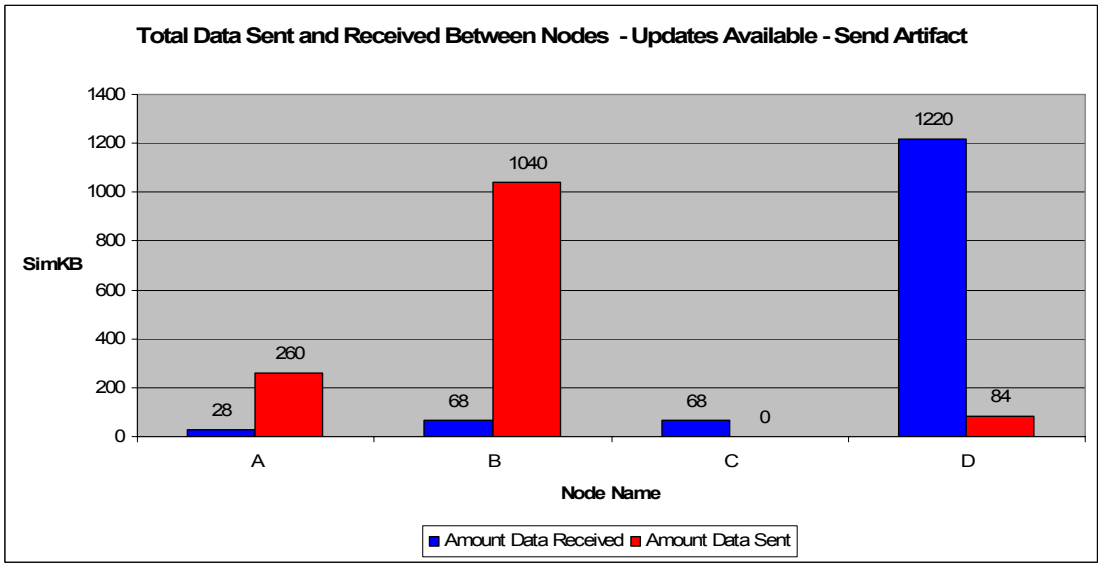


Figure 34 : Data traffic between nodes where artifact updates exist and the artifact is sent

In the case where artifact updates do not exist, the same amount of data is sent between the nodes. Figure 35 shows that even though there is no update that is made,

the artifact *"Simple.cs"* is transferred from *Node A* and the artifact *"Universe.jpg"* transferred from *Node B*.

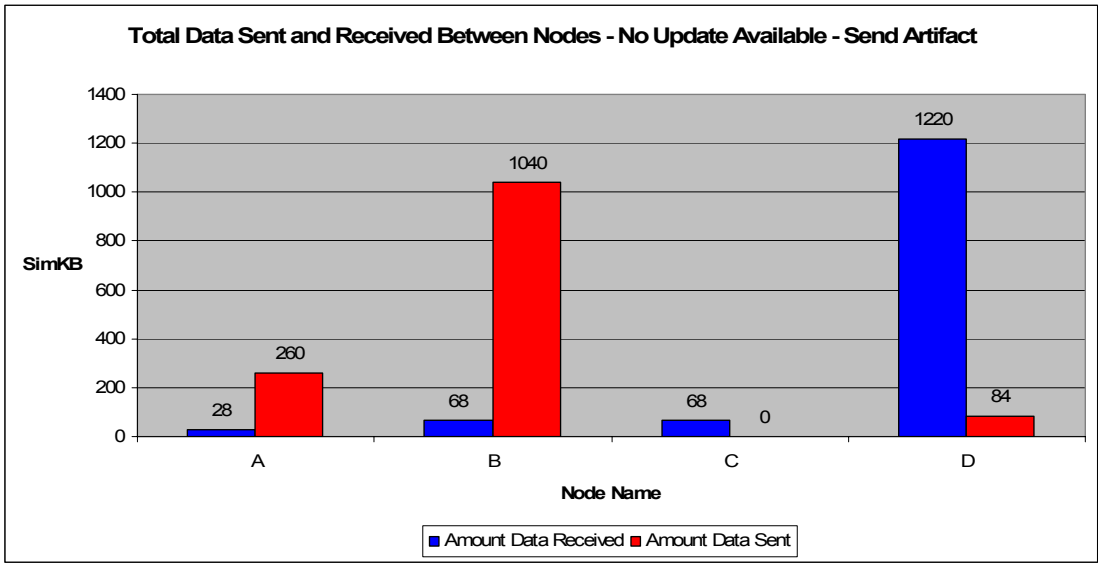


Figure 35 : Data traffic between nodes where artifact updates do not exist and the artifact is sent

There is an overhead caused by the requestor nodes on the collaborator nodes in the case where no updates exist. The artifacts are transported over the network from the collaborator nodes to the requestor nodes, only to be discarded by the requestor node.

It should be noted that *Node C* has no overhead. It received the request, but since it had neither of the artifacts, nothing has been transferred back to the requestor.

4.1.3. Sending of artifact details

We need some way to reduce the overhead of the nodes if no updates exist. We take another look at Figure 33.

If an artifact does not exist on a node, then we can allow the artifacts to be sent to the requestor node. Furthermore, if the requestor node knows who the owner of the artifact is, the requestor could request the artifact from the owner initially, and then could update the artifact thereafter using the *best effort* approach.

If an artifact exists on the node, then the details of the artifact on the requestor could be sent as part of the shopping list to all collaborators. Only those collaborators whom have a newer version of the artifact will respond to the shopping list.

The requestor has two options. If the artifact is small, the requestor could request that any artifact newer than the version that exists on his node, be sent to him. If the size is unknown, or rather large, the requestor could request that the details of the artifact that exist on the collaborators node be sent to him. Once the artifact details of the collaborator nodes reaches the requestor, the requestor can make an informed decision on which artifact he would prefer, and request that artifact. Figure 36 illustrated this concept.

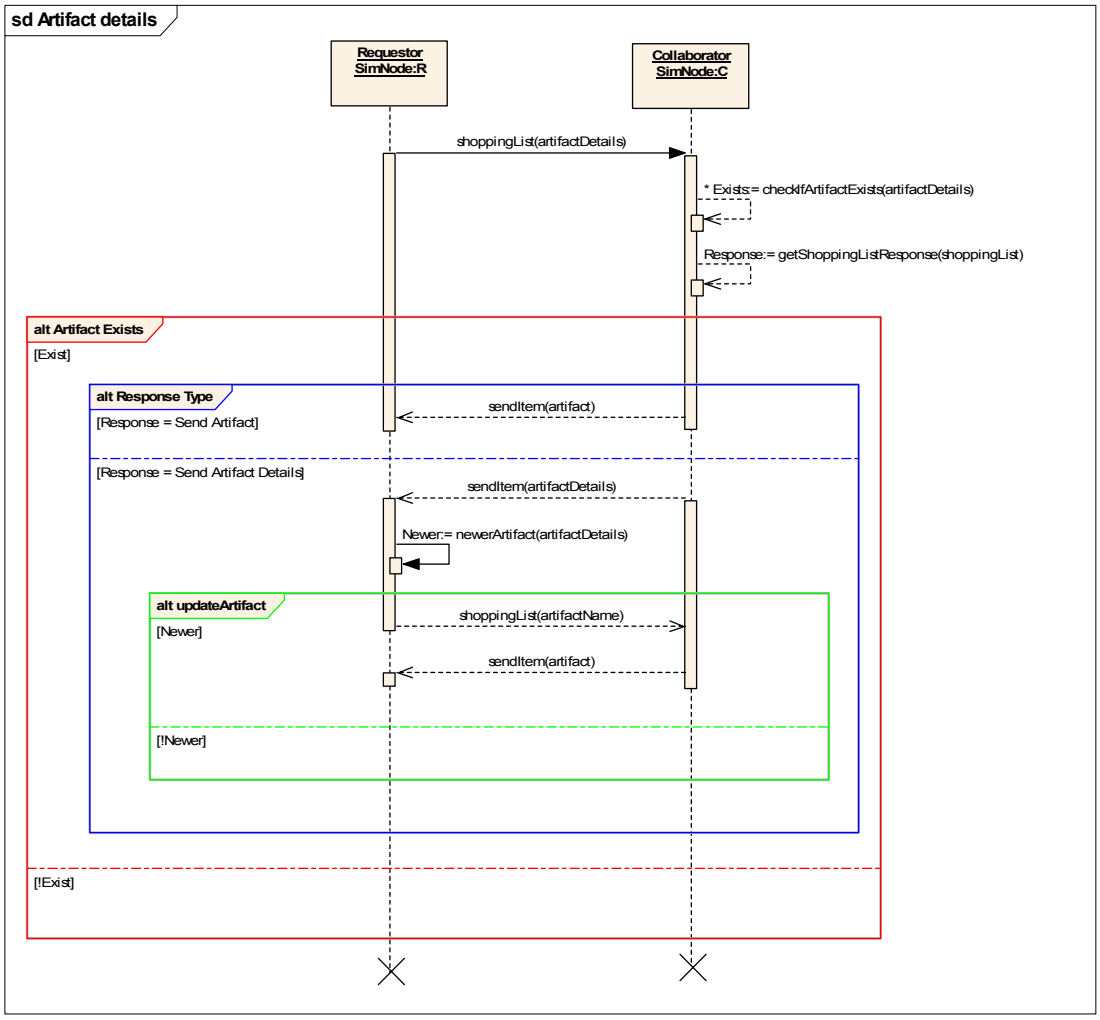


Figure 36 : Send Artifact Details

We notice that there are more interactions between the nodes as compared to sending the artifact only (see section 4.1.2). This is due to the sending of artifact details between the nodes. Figure 37 shows the contents of the artifact details message.

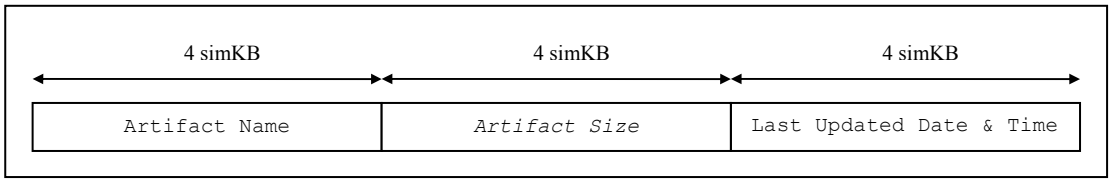


Figure 37 : Artifact details structure

But will these interactions hold a heavier or lighter overhead on the collaborator nodes?

If updates exist on other nodes, then the total data transferred from other nodes are as in Figure 38. *Node A* transferred the updated version of *"Simple.cs"*. *Node B* transferred the updated version of *"Universe.jpg"*. We compare these figures to that of Figure 34. There is an increase in the amount of data that is transferred between the nodes. This is expected due to the sending of the artifacts details message.

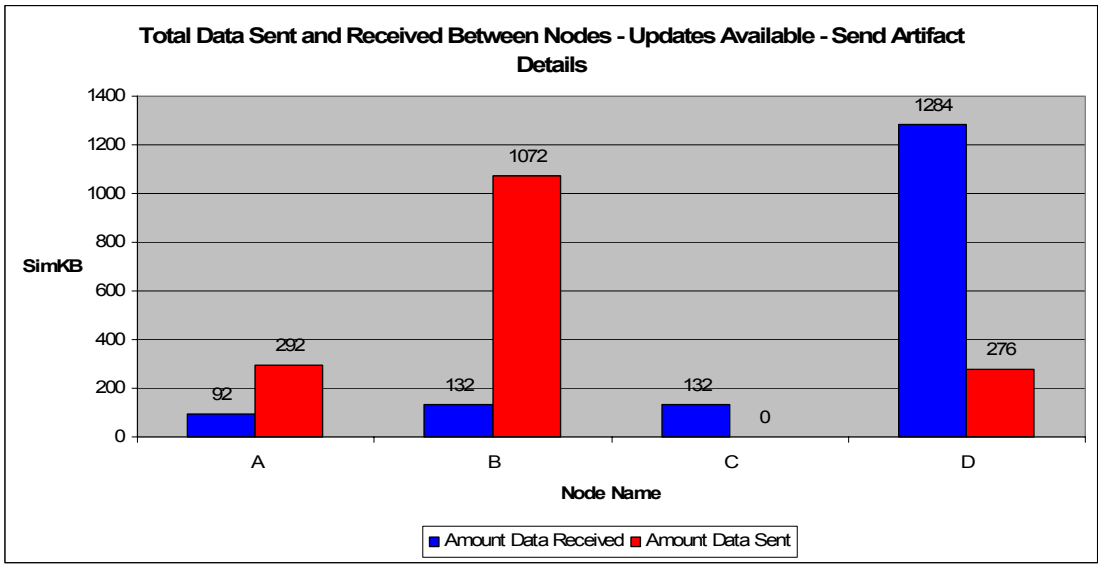


Figure 38 : Data traffic between nodes where artifact updates exist and the artifact details are sent

In the case were artifact updates do not exist, a reduced amount of data is transferred between the nodes. Figure 39 demonstrates the reduced overhead on the collaborators when the requestor has the latest version of the artifact initially. Although the artifact

"Simple.cs" exists on *Node A* and the artifact *"Universe.jpg"* exists on *Node B*, they are older than the version that exists on the requestor node, *Node D*. They are therefore not transferred as they ultimately would be discarded on *Node D*.

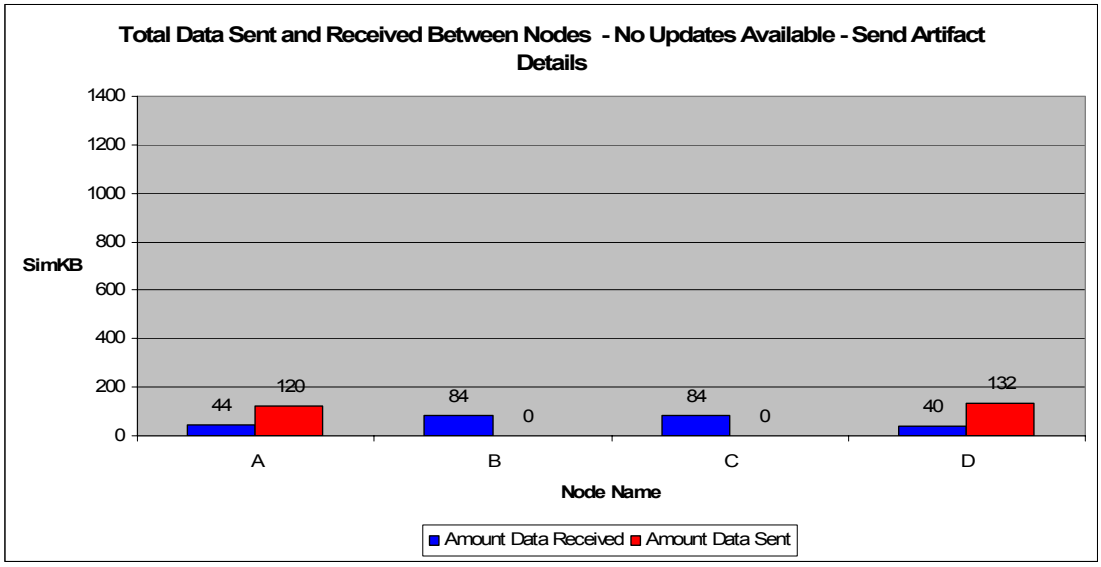


Figure 39 : Data traffic between nodes where artifact updates do not exist and the artifact details are sent

It should be noted that *Node C* has no overhead in this case as well. It received the request, but since it had neither of the artifacts, nothing has been transferred back to the requestor.

The problem now exists on when is the artifact large enough to be sent across the network? If the internet connection is slow, we do not want to receive big files unnecessarily. But could we reduce the amount of interactions and implement some type of configuration on how large, *is large*?

4.1.4. Introducing the artifact size limit

We are faced with a dilemma, where artifacts are easily transportable over excellent internet connections, and the node has an abundance of processing power; and weak connections on nodes with less processing power, that struggle to download the same artifacts.

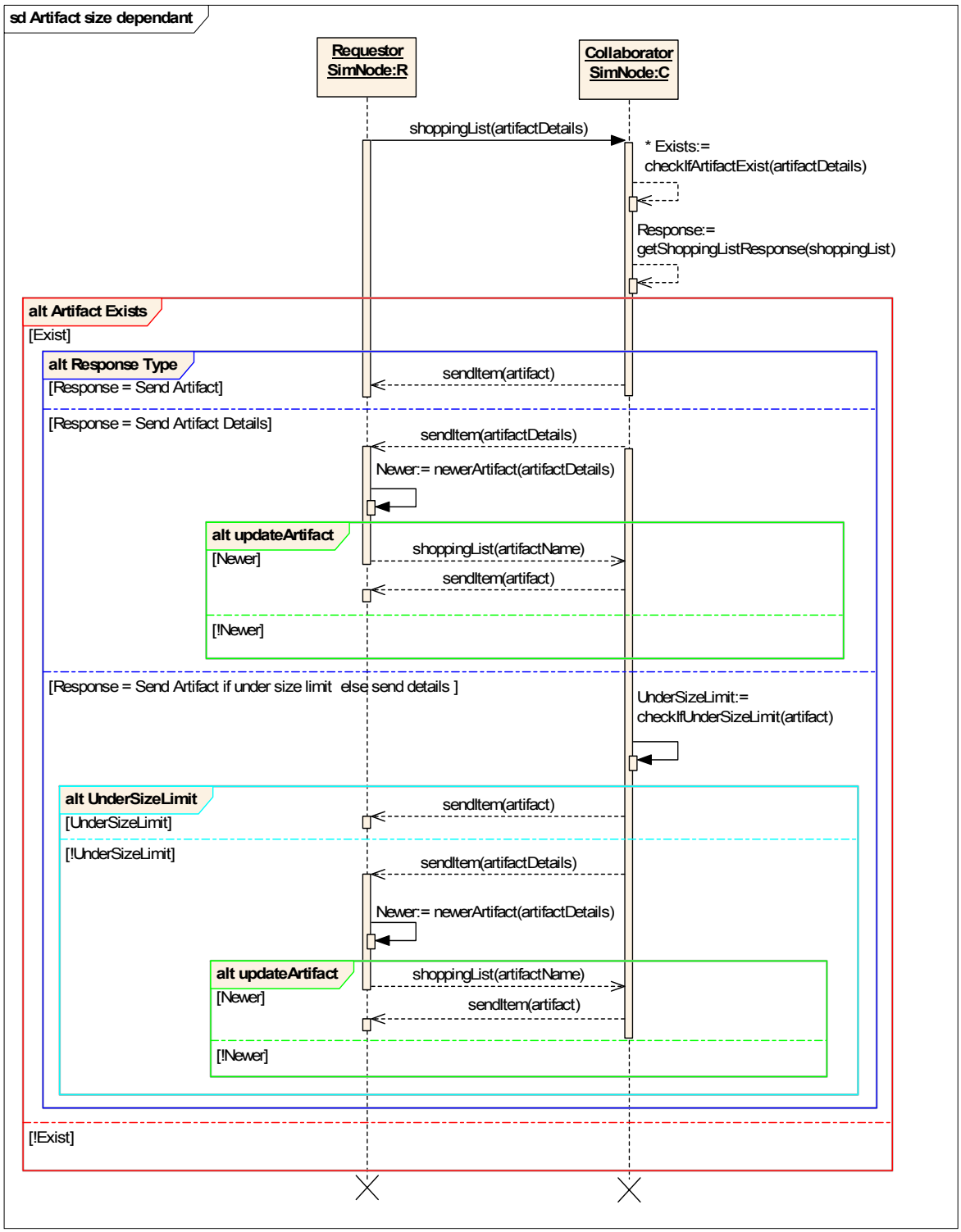


Figure 40 : Combined Response

We implement functionality to customize what a node considers a *large* artifact. For this scenario, we choose a size limit of *400 simKB*. If an artifact is less than this size, it will be transported to the requestor node without intervention. If the artifact is larger than this size, the details of this artifact are sent back to the requestor. The requestor

can make an informed decision of whether the artifact is needed. Figure 40 shows the sequence of events for each of these response types.

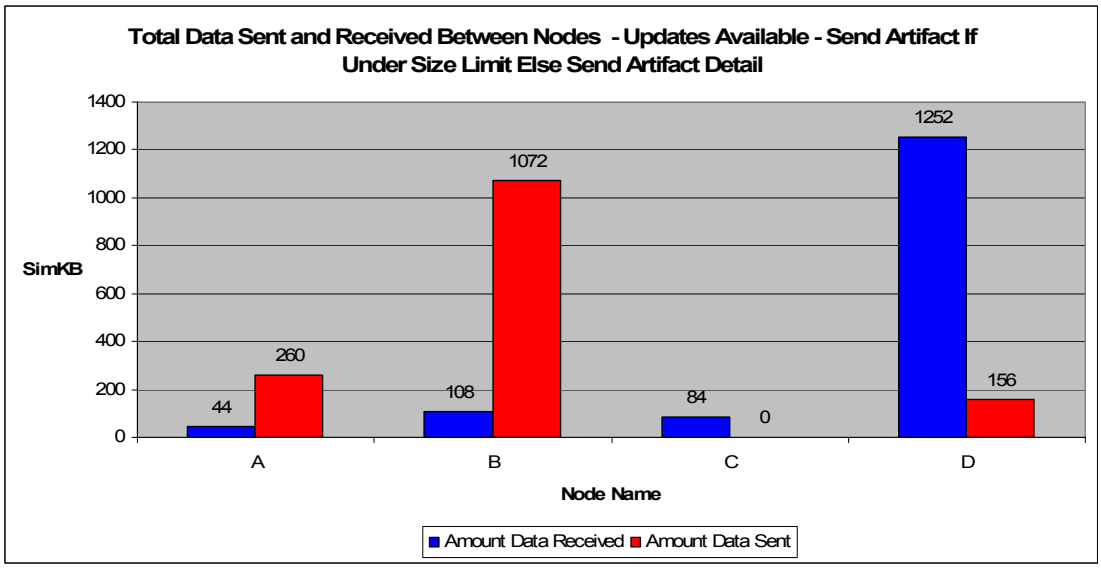


Figure 41 : Data traffic between nodes where artifact updates exist and the artifact sent dependant on size

If updates exist on other nodes, then the total data transferred from other nodes are as in Figure 41. *Node A* transferred the updated version of *"Simple.cs"*. *Node B* has transferred the updated version of *"Universe.jpg"*. We compare these figures to that of Figure 34 and Figure 38. The values are between the comparative figures. There is an increase in the amount of data that is transferred between the nodes. Specifically, an extra message is transferred and received from *Node B*. This is since the artifact *"Universe.jpg"* is over the size limit.

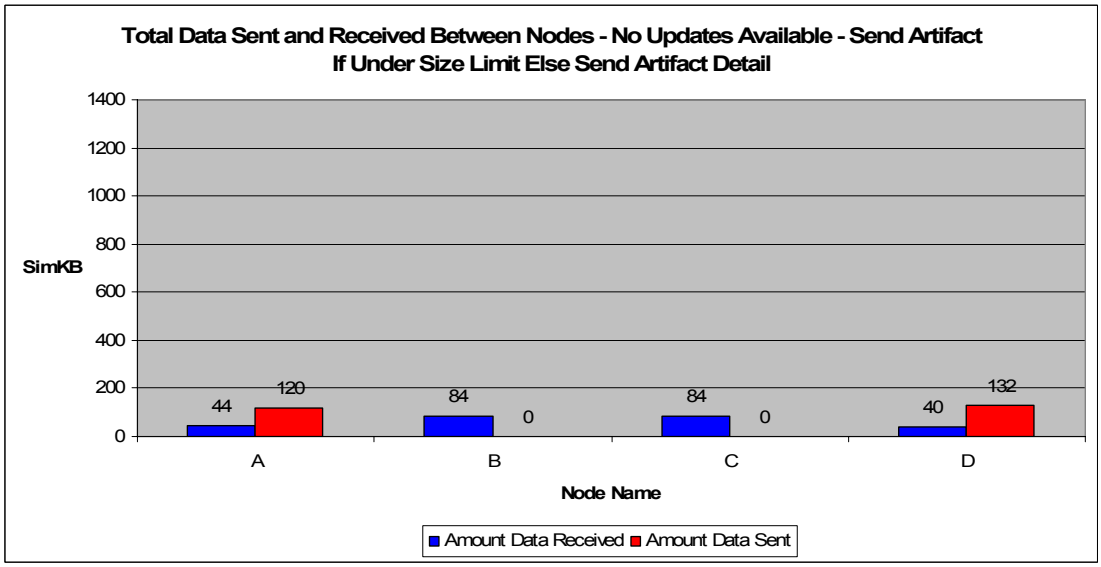


Figure 42 : Data traffic between nodes where artifact updates do not exist and the artifact sent dependant on size

In the case were artifact updates do not exist, the same values of Figure 39 are achieved. Figure 42 shows the results of this option.

4.1.5. Conclusion

If the artifact does not exist on the requestor node, we need to get the artifact from collaborator nodes. All artifacts will be sent to the requestor including older versions and duplicates of a new version. This causes an overhead on the collaborator nodes. If the artifact size is relatively small, the collaborators will include a relatively small and possibly unnoticeable overhead. If the artifact size is relatively large, the overhead on the collaborator nodes will be large and even more evident if the node has a weak or slow internet connection.

But in the case where the artifact already exists on the requestor node, we can make use of the *artifact details* option. This solution shows that there is far less overhead on the nodes if the artifact that is present on the collaborators node is older than that of the requestor's node. Furthermore we could include some logic which allows the system to make a decision to send the artifact based on its size and modification date. This is described in section 4.1.4.

Since the use of artifact details reduces overhead on other collaborators nodes, it is a viable solution that should be included in the Nomad protocol. Since artifact details can only exist if the artifact exists, we offer the suggestion to make use of dummy artifacts which will have the timestamp of the creation of the project. In that way, we could eliminate the need to send an artifact, especially larger artifacts, which will ultimately be discarded on the requestor's node, due to the artifact being a later or same version that exists on the requestor's node.

4.2. Simple disconnection scenario

4.2.1. Scenario setup and description

This scenario is made up of the following nodes as described in Table 3:

Node name	Node Type
A	TYPE1
B	TYPE2
C	TYPE3
D	TYPE2

Table 3 : Simple disconnection scenario setup

A visual description from the NPS is shown in Figure 43.

Figure 43 : NPS Simple Disconnection Scenario

This simulation is used to show the effect of different types of devices in a *disconnected* environment. There are *no* artifacts that are transferred. Nodes change from being online to offline following a specific order. Figure 44 shows the events that take place in this scenario. At event 2, Node B goes offline. It becomes available online at event 4. At this point it sends an online notification to the other online nodes, namely *Node A*, *Node C* and *Node D.* At event 6, *Node B*, *Node C* and *Node D* go offline.

Node B returns to being online at event 8. It has a 50% chance of having the same IP address. It sends an online notification to the only online node, *Node A.* Since *Node C* and *Node D* are offline and *Node B* is a TYPE 2 node, an email has to be sent by *Node*

B, so if those nodes become online, they know on which IP address *Node B* is available on.

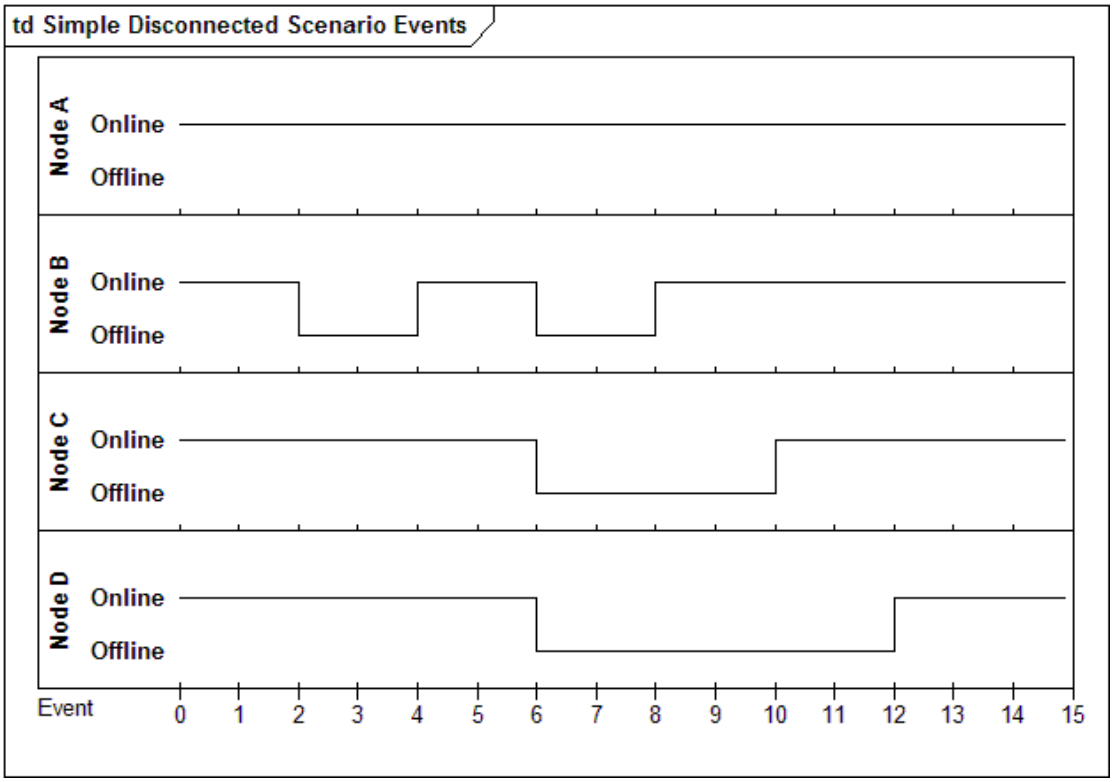


Figure 44 : Simple disconnected scenario timeline

Event 10 denotes *Node C* going online. The IP address of *Node C* will be changed to a IP different from the one it had the previous time it was online. It sends an online notification to the online nodes, namely, *Node A* and *Node B*. At this point, *Node D* is still offline. Since *Node C* is a TYPE 3 node, and a characteristic of this type of node is that it will be online for a very long period of time. If *user interaction* is switched on, the user is posed with the question of whether to send an email to *Node D*. This option allows the node to send its latest IP address in an email. This option would be needed if *Node C* is going to be online for a long period of time.

Node D goes online at event 13. It has a 50% chance of changing its IP address. At this point it sends an online notification to all online nodes, namely *Node A*, *Node B* and *Node C*.

We now analyze and evaluate the results gathered from this scenario.

4.2.2. Results

We show that the use of an online notification works well in a disconnected scenario. Figure 45 shows the amount of data transferred during this scenario.

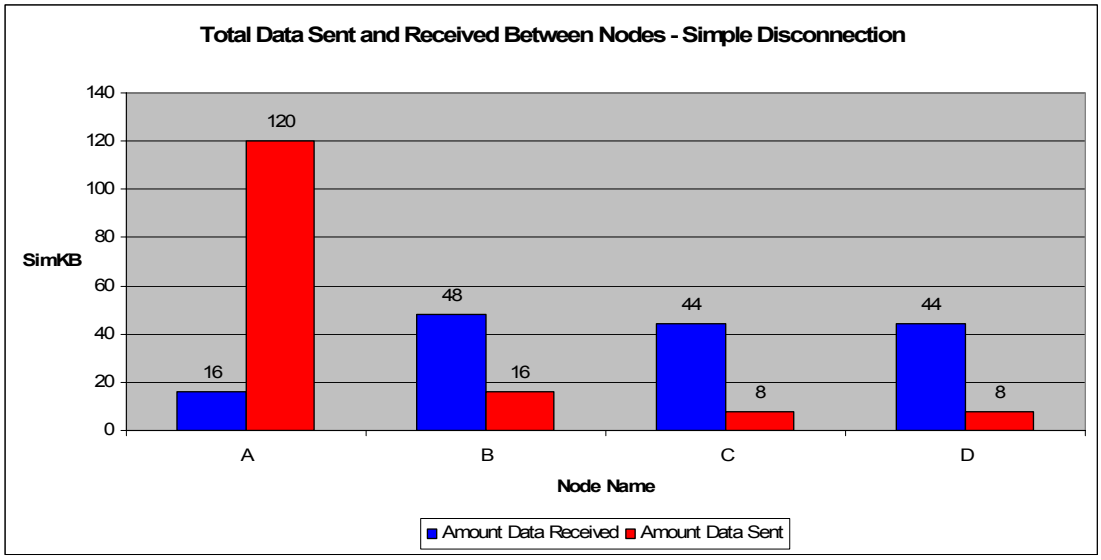


Figure 45 : Data transferred in the simple disconnected scenario

Figure 46 illustrates the counters of each node. We now take a more detailed look at these figures. *Node A* received 4 online notifications, which totals to 16 simKB. It did not transfer data due to the nodes disconnections. The 120 simKB is due to sending project details to the three collaborator nodes at the start of the simulation.

Node B sent four and received two notifications. The total of 16 simKB has been sent and 8 simKB received on the node due to its own disconnection and the disconnection of other nodes in the scenario. The 40 simKB received is due to the project details received upon startup of the project. The same applies to *Node C* and *Node D*.

Node C and Node D both received one and sent two online notifications. The total of 8 simKB has been sent and 4 simKB received on the nodes due to its own disconnection and the disconnection of other nodes in the scenario.

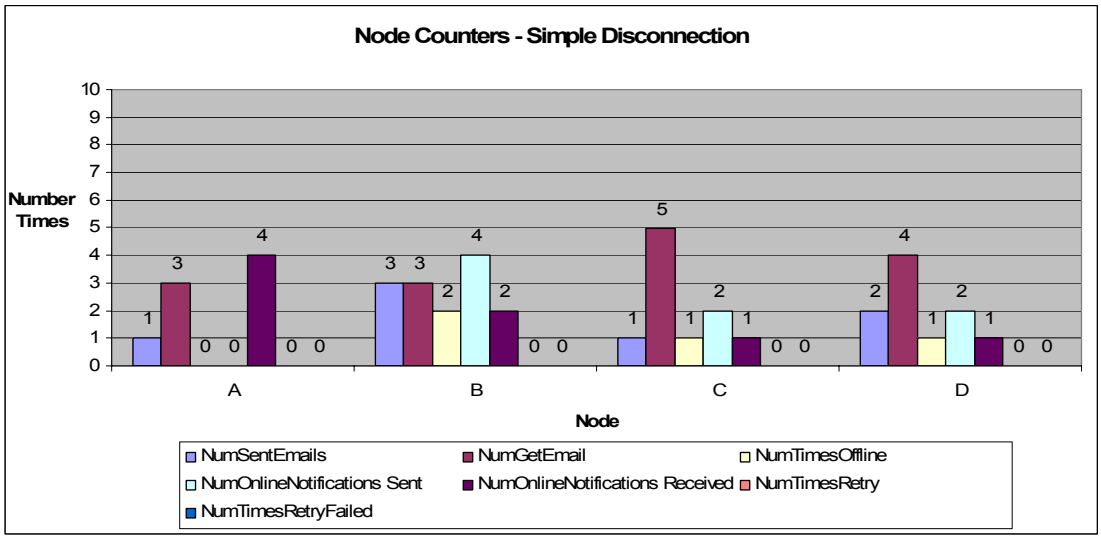


Figure 46 : Node counters for simple disconnected scenario

An important aspect for the nodes is the amount of email sent by the online nodes to the offline nodes on becoming available with a new IP address. *Node C* received two emails since there were two nodes that were assigned new IP addresses during the time *Node C* was offline.

4.2.3. Conclusion

It was seen that the approach of the protocol is sufficient for the devices that the protocol intends to cater for. A small, modest overhead is placed on the online nodes in terms of data transferred and emails received. The amount of online notifications is proportional to the amount of times a node becomes available online.

The protocol caters for devices that will be online for short periods of time, and if necessary, user interaction can play the deciding factor on whether a collaborator should make its presence known in the community.

The protocol makes use of the email framework to keep nodes in contact in cases that nodes come online with IP addresses that are different to those that other collaborators are aware of.

4.3. Desired Effort Levels

4.3.1. Scenario setup and description

This scenario is made up of the following nodes as described in Table 4:

Node name	Node Type	Propagation Node
A	TYPE1	Yes
B	TYPE2	No
C	TYPE3	No
D	TYPE2	No
E	TYPE1	No
F	TYPE2	No

Table 4 : Desired effort levels scenario setup

A visual description from the NPS is shown in Figure 47.

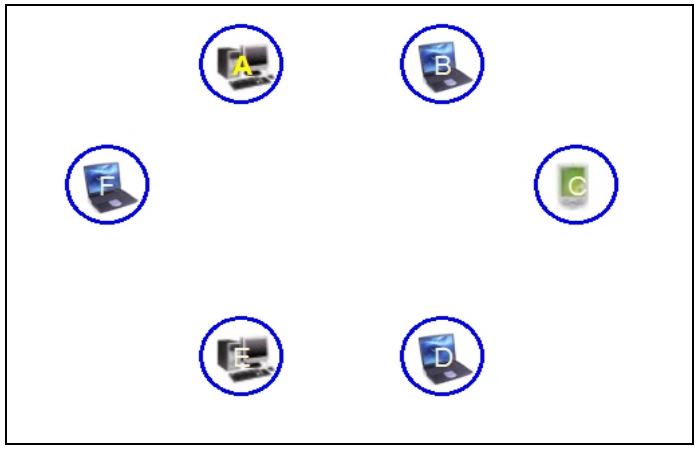


Figure 47 : NPS desired effort level scenario

This scenario compares the different effort levels of retrieving an artifact from the community. The NPS illustrates these approaches as

- Best effort approach,
- Require item approach, and
- Require item under any condition.

We discuss each approach later in this section. Figure 48 shows the sequence of events for each of the approaches when transferring an item between nodes.

```
public void transferItem(int from_node_id,
                         int to_node_id,
                         SimTransportItem item)
{
  enum result {SUCCESSFUL, FAILED};
  E_DESIRED_EFFORT_LEVEL effort;

  /* transportItem via management queue*/
  result = transportItem(from_node_id, to_node_id, item);

    if (result == SUCCESSFUL)
    {
      if(item.type_of_request ==
↳ E_TYPE_OF_REQUEST.FORWARDED_REQUEST_RESPOND_IF_REQUEST_FUFILLED)
      {
        /* Get the original request */
        SimTransportItem embedded_transport_item =
↳ item.get_forwarded_request();

          /* Create a SimRequestFufilled item */
          SimRequestFufilled req_fufilled = SimRequestFufilled
↳(embedded_transport_item.FromNode,this.NodeId item.ForwardRequestID);

          /* State it was successful */
          req_fufilled.request_fufilled = E_REQUEST_FUFILLED.YES;

          /* Add the request fulfilled item via the management queue */
          add_to_items_to_be_transferred_list(req_fufilled);
      }
      else if (item.type_of_request ==
↳ E_TYPE_OF_REQUEST.FORWARDED_REQUEST_NO_FURTHER_INTERACTION_NEEDED)
      {
        /* No response required */
      }//end else

      /* remove item from management queue*/
      deQueue(item);
    }
    else /* result == FAILED*/
    {
      /* Get the desired effort level from the item */
        effort = item.EffortLevel;

        /* ---------------------- BEST EFFORT  ---------------------- */
        if (effort == E_DESIRED_EFFORT_LEVEL.BEST_EFFORT)
        {
          /* remove item from management queue*/
          deQueue(item);
        }
        /* ---- REQUIRE ITEM & REQUIRE_ITEM_UNDER_ANY_CONDITION ---- */
        else if ((effort == E_DESIRED_EFFORT_LEVEL.REQUIRE_ITEM) ||
        (effort == E_DESIRED_EFFORT_LEVEL.REQUIRE_ITEM_UNDER_ANY_CONDITION))
        {
          if(effort ==
↳E_DESIRED_EFFORT_LEVEL.REQUIRE_ITEM_UNDER_ANY_CONDITION)
          {
            /* reset the number of retries */
            item.NumberOfRetries = 0;
          }//end if

          /* check if we are on the propagation node */
          if (PropagationNode == E_PROPAGATION_NODE.YES)
```

```
          {
            if(item.type_of_request ==
↳E_TYPE_OF_REQUEST.FORWARDED_REQUEST_RESPOND_IF_REQUEST_FUFILLED)
            {
              /* Get the original request */
               SimTransportItem embedded_transport_item =
↳item.get_forwarded_request();

               /* Create a SimRequestFufilled item */
               SimRequestFufilled req_fufilled = SimRequestFufilled
↳(embedded_transport_item.FromNode, this.NodeId, item.ForwardRequestID);

               /* State it was not successful */
               req_fufilled.request_fufilled = E_REQUEST_FUFILLED.NO;

               /* Add the request fulfilled item via the management queue*/
               add_to_items_to_be_transferred_list(req_fufilled);
            }
            else if (item.type_of_request ==
↳ E_TYPE_OF_REQUEST.FORWARDED_REQUEST_NO_FURTHER_INTERACTION_NEEDED)
            {
              /* No response required */
            }//end else
          }
          else /* not a propagation node */
          {
            if (sim_internet.check_user_interaction_status() == ENABLED)
            {
              /* dialog box requiring YES or NO response*/
              choice =  promptUser("Send request to Propagation nodes?");
              if (choice == Yes)
              {
                /* send this item to all propagation nodes */
                send_item_to_all_propagation_nodes(item);
              }//end if
            }
            else //automated
            {
              /* send this item to all propagation nodes */
              send_item_to_all_propagation_nodes(item);
            }//end else - user enabled
          }//end else - prop node

          /* dequeue or leave on queue depending on effort level*/
          if (effort == E_DESIRED_EFFORT_LEVEL.REQUIRE_ITEM)
          {
            /* remove item from management queue*/
            deQueue(item);
          }
          else if (effort ==
↳ E_DESIRED_EFFORT_LEVEL.REQUIRE_ITEM_UNDER_ANY_CONDITION)
          {
            /* Leave item on queue - Allows to retry */
          }//end else
        }//end else - effort level
    }//end else - failed
}//end transferItem()
```

Figure 48 : Pseudo code for the NPS transferItem function

We show that although more recently updated artifacts exist on other nodes, the best approach method only gathers needed artifacts, defined in a shopping list, from nodes that are currently online.

We introduce the propagation node, defined as *Node A* in the scenario and depicted with a yellow bold symbol as in Figure 47. Although the propagation node has existed in the previous scenarios, it plays a vital role in this scenario.

Table 5 shows the spread of artifacts that exist on each node. The use of *version* in this instance relates to the latest modification time, or the `last_update_date_time` attribute as defined in section 3.1.2.2.

Artifact name	**Initial artifact size (*simKB*)**	**Node A**	**Node B**	**Node C**	**Node D**	**Node E**	**Node F**
simple.cs	100	Ver2	-	-	Ver1	-	Ver3
universe.jpg	1000	-	Ver2	-	Ver1	-	Ver3
flowchart.jpg	2000	-	-	-	-	-	Ver1

Table 5 : Artifact versions on each node

Node F has the latest versions of all three artifacts, and is the only one in the community that has the artifact *"flowchart.jpg"*.

When the simulation starts, the project is setup amongst the nodes. *Node D* then creates two artifacts, namely *"simple.cs"* and *"universe.jpg"*. *Node A* and *Node B* then create later versions of "*simple.cs"* and *"universe.jpg"* respectively.

Node F goes offline, but proceeds to create the latest versions of all three artifacts.
At the same time *Node B* and *Node F* go offline.

Node D then creates a shopping list which it sends to all available nodes. Figure 49 and Figure 55 illustrate the events during this scenario. In each of the following cases, the *effort level* depicts the effort of the community to fulfill the shopping list. If needed, it makes it possible that the latest artifacts reach the requesting node, *Node D*.

We now take a look at each of these methods and give the artifact list of the requesting node, *Node D*, at the start and end of simulation.

4.3.2. Best effort approach

The best-effort approach will be the most used desired approach of Nomad collaborators. This approach gathers a community wide view of all collaborators that are online. If a requesting node sends out a shopping list and a collaborator node is offline or not available to the community, the request to that node is discarded. Requests are only sent to online nodes.

There are two shopping lists that are sent one after the other. The first contains the details of the artifacts *"simple.cs"* and *"universe.jpg"*. Since the artifact *"flowchart.jpg"* does not exist on *Node D*, we cannot use the artifact details, but instead send a shopping list with the *artifact name* only. See section 4.1 for more information. This is denoted by *D2* in Figure 49.

Table 6 show the meta-data of the artifacts that exist on *Node D* before the shopping list is sent out. We later see how the artifacts are updated.

Artifact Name	*simple.cs*	*universe.jpg*	*flowchart.jpg*
Size	100	1000	-
Created	2/10/2006 10:30:36 PM625	2/10/2006 10:30:36 PM625	-
Last update	2/10/2006 10:30:36 PM625	2/10/2006 10:30:36 PM625	-
Author	D	D	-
Origin node	4	4	-
Last Author Update	D	D	-
Last Artifact Location	4	4	-

Table 6 : Artifact list on Node D before sending shopping list

Figure 49 shows an event related sequence of the NPS best effort approach scenario. We see that at event 2, *Node B*, *Node C* and *Node F* go offline. Soon after, *Node D*

sends out a best effort shopping list. The online nodes, *Node A* and *Node E* receive the shopping list. *Node A* responds with an updated version of *"simple.cs"* and *Node D* updates its artifact list.

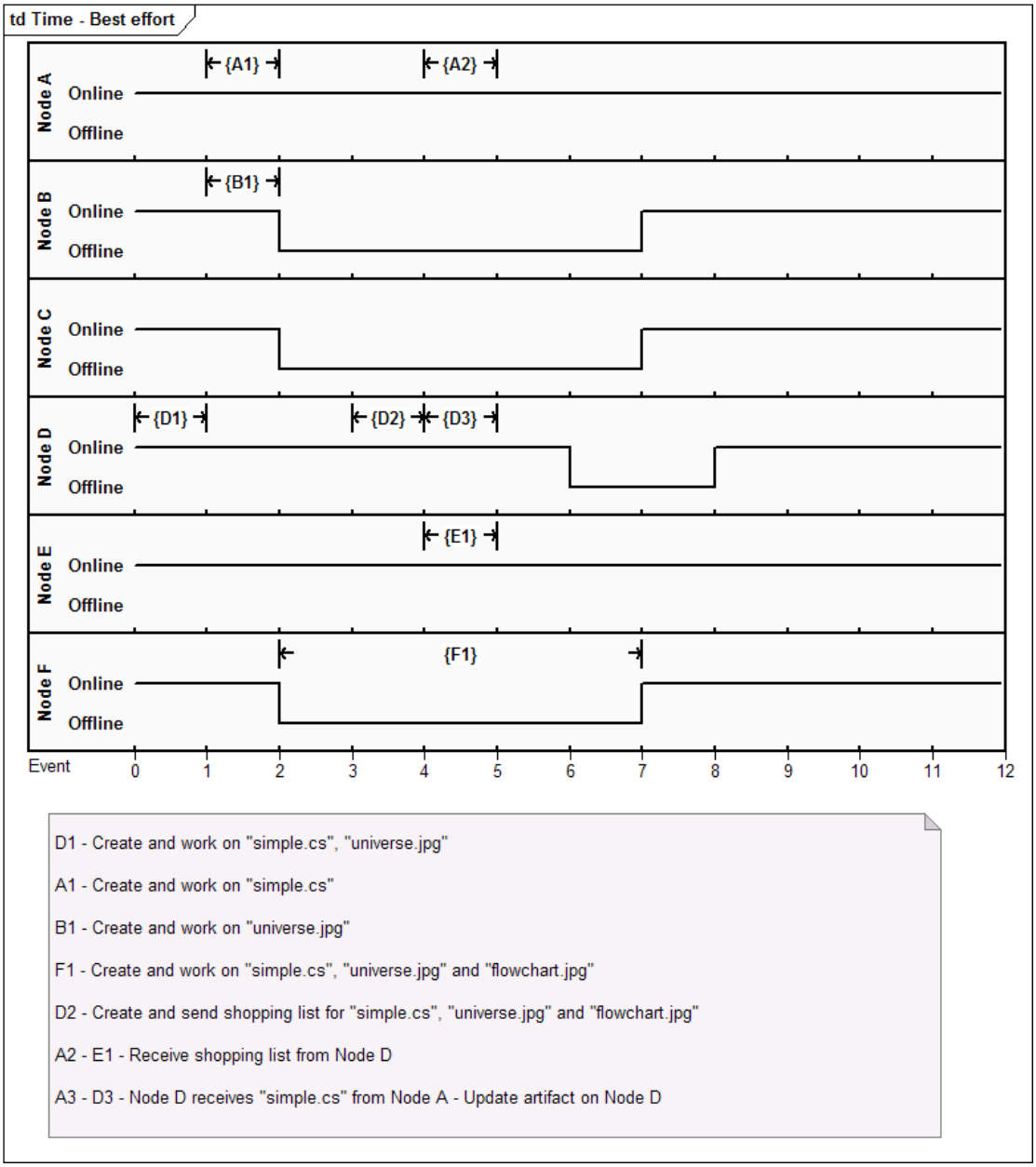


Figure 49 : Best approach scenario timeline

Table 7 shows the artifact list after the update has been made. We note that the meta-data for *"simple.cs"* now has the attributes of the artifact on *Node A*. The *last author update* and *last artifact location* attributes of *"simple.cs"* denote that this artifact has been updated from *Node A*. The artifact *"flowchart.jpg"* has not been found on any online nodes, and therefore has not been received.

Artifact Name	*simple.cs*	*universe.jpg*	*flowchart.jpg*
Size	120	1000	-
Created	2/10/2006 10:30:36 PM625	2/10/2006 10:30:36 PM625	-
Last update	2/10/2006 10:30:39 PM15	2/10/2006 10:30:36 PM625	-
Author	D	D	-
Origin node	4	4	-
Last Author Update	A	D	-
Last Artifact Location	1	4	-

Table 7 : Artifact list on Node D after update

The data transferred between the nodes are presented in Figure 50. The offline nodes do not send or receive data due to the shopping list being sent. We note for reasons stated above, two shopping lists need to be sent out in this case.

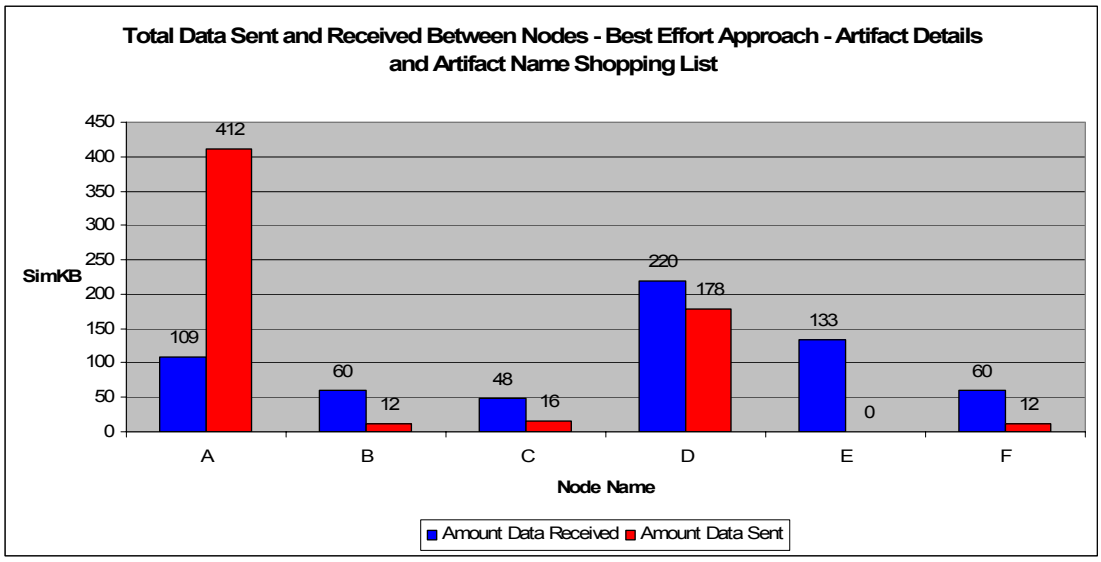


Figure 50 : Data transferred and received between nodes - Best Effort Approach

Figure 51 shows the counters of each node in the simulation. We note that *Node D* has six *failed retries* due to the two shopping lists per three offline nodes.

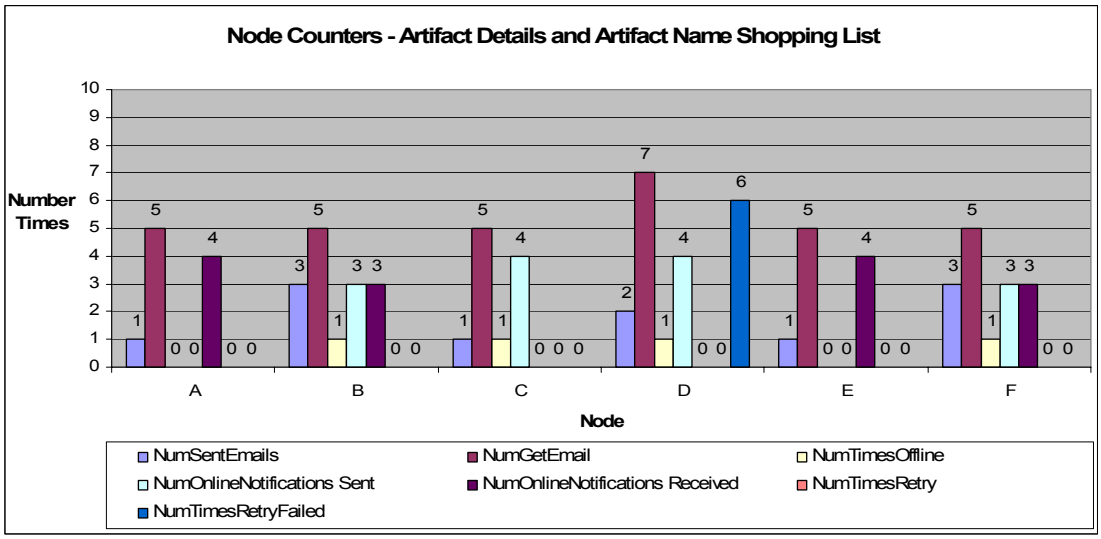


Figure 51 : Node Counters Best Effort approach

Figure 52 clearly defines the retry counts per node. We see that Node D is the only node that suffers from a retry count.

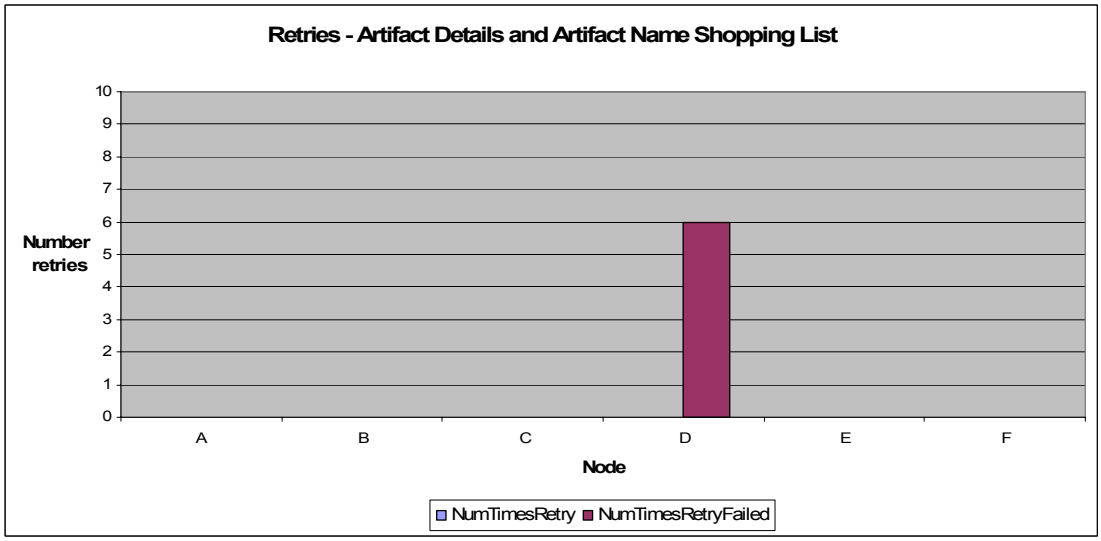


Figure 52 : Retries per node in Best Effort approach

We could have easily sent a single shopping list with all three artifact names. Figure 53 illustrates the data transferred and received between nodes with a single shopping list. We notice that the amount of data on the offline nodes is the same as in Figure 50, but the overhead of data on the online nodes are less.

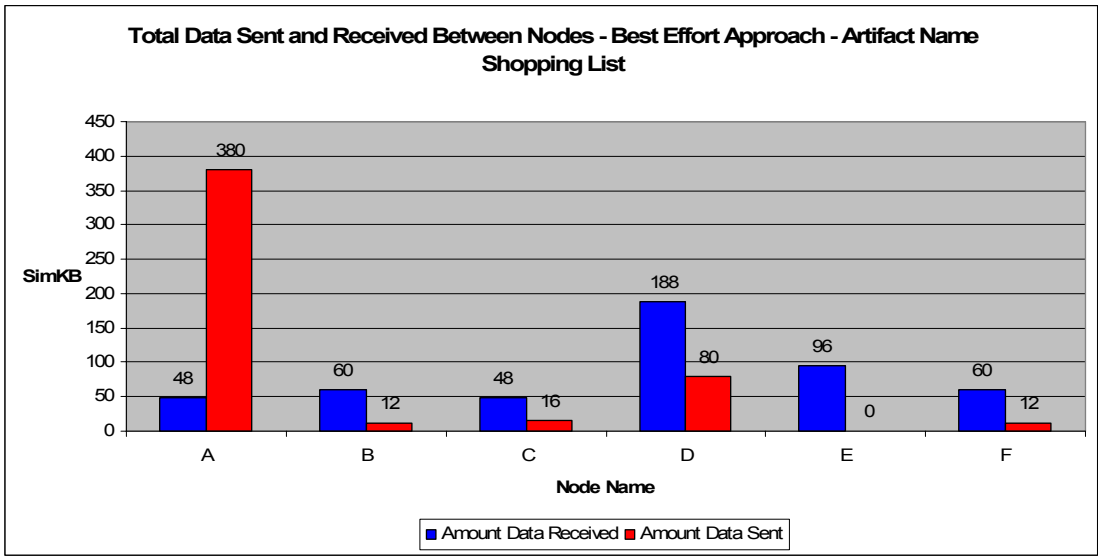


Figure 53 : Data transferred and received - Best effort with single shopping list

We take note that the number of retries on the requesting node, *Node D*, is halved to three since that is only a single shopping list being sent. This is illustrated in Figure 54.

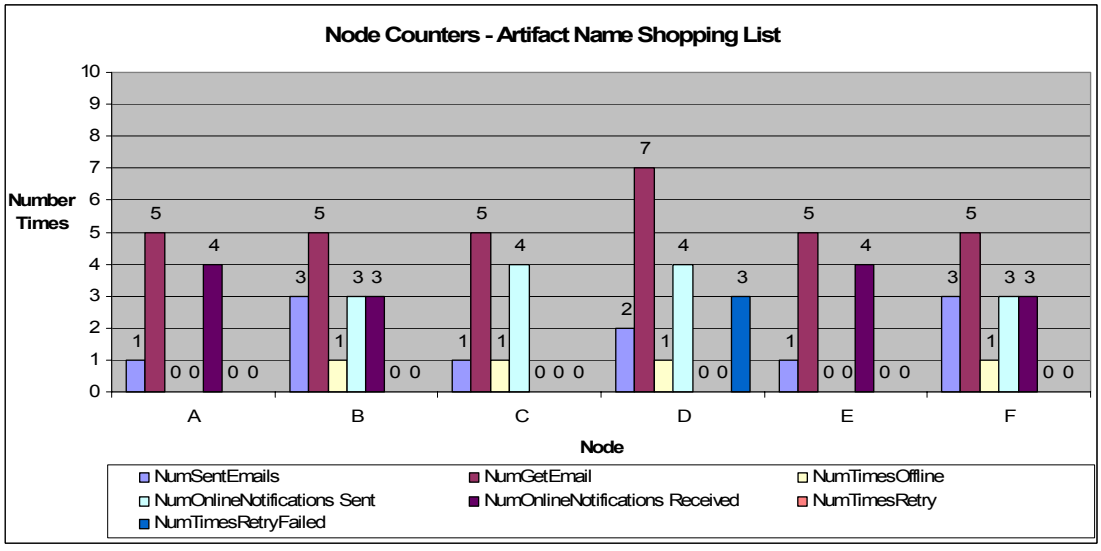


Figure 54 : Node Counters - best effort with single shopping list

In this particular case, the artifacts transferred, namely "*simple.cs*" from *Node A*, is a more updated version of the artifact. If however, the artifact was the same or earlier version of the artifact, there would be unnecessary transferring of data between the nodes. This is therefore a disadvantage of a single shopping list. This is more apparent in the following scenario.

4.3.3. Require item

The *require item* approach will be the used if the artifacts exist on nodes that are currently offline. This approach gathers a community wide view of all collaborators that are online and later gathers required artifacts from nodes that were offline at the time the shopping list was sent. We now introduce the *propagation node* and the *request fulfilled* reply to the scenario. The propagation node and request fulfilled reply are further discussed in section 3.1.2.3.

If a requesting node sends out a shopping list and a collaborator node is offline or not available to the community, the request to that node is sent to a *propagation node*. In this scenario, *Node A* acts as a propagation node. Once these nodes are online, the requests are forwarded to them. If the item was a `FORWARDED_REQUEST_RESPOND_IF_REQUEST_FULFILLED` item, the propagation node sends the requestor node, *Node D*; a request fulfilled denoting the success of forwarding this item.

If the nodes do not become available to the community after a period of time, and the retry count on the propagation node is reached, the forwarded item is discarded. If the item was a `FORWARDED_REQUEST_RESPOND_IF_REQUEST_FULFILLED` item, the propagation node sends the requestor node, *Node D*; a request fulfilled denoting the failure of forwarding this item.

If the item was a `FORWARDED_REQUEST_NO_FURTHER_INTERACTION_NEEDED`, in both cases, the propagation node does not respond to the requesting node with the success or failure of forwarding this item.

Table 8 show the meta-data of the artifacts that exist on Node D before the shopping list is sent out. We later see how the artifacts are updated.

Artifact Name	*simple.cs*	*universe.jpg*	*flowchart.jpg*
Size	100	1000	-
Created	2/14/2006 11:39:18 PM703	2/14/2006 11:39:18 PM703	-
Last update	2/14/2006 11:39:18 PM703	2/14/2006 11:39:18 PM703	-
Author	D	D	-
Origin node	4	4	-
Last Author Update	D	D	-
Last Artifact Location	4	4	-

Table 8 : Artifact list on Node D before sending shopping list

As in the previous scenario, there are two shopping lists that are sent one after the other. The first contains the details of the artifacts *"simple.cs"* and *"universe.jpg"*. Since the artifact *"flowchart.jpg"* does not exist on *Node D*, we cannot use the artifact details, but instead send a shopping list with the *artifact name* only.

Figure 55 shows an event related sequence of the NPS "*require item"* approach scenario. We see that at event 2, *Node B*, *Node C* and *Node F* go offline. Soon after, *Node D* sends out a best effort shopping list. The online nodes, *Node A* and *Node E* receive the shopping list. *Node A* responds with an updated version of *"simple.cs"* and *Node D* updates its artifact list.

After a number of retries, *Node D* sends a forward request containing the shopping list to all propagation nodes. In this scenario, *Node A* is the only a propagation node in the community. After the requests are forwarded, *Node D* goes offline.

During this time, the three previously offline nodes go online. These are *Node B*, *Node C* and *Node F*. Once these nodes send an online notification to all online nodes, the propagation node sends the hosted forwarded request from *Node D* to each node. If the IP addresses of these nodes have changed, and since *Node D* is at that time offline, an email with the new IP address of these nodes is sent to *Node D*.

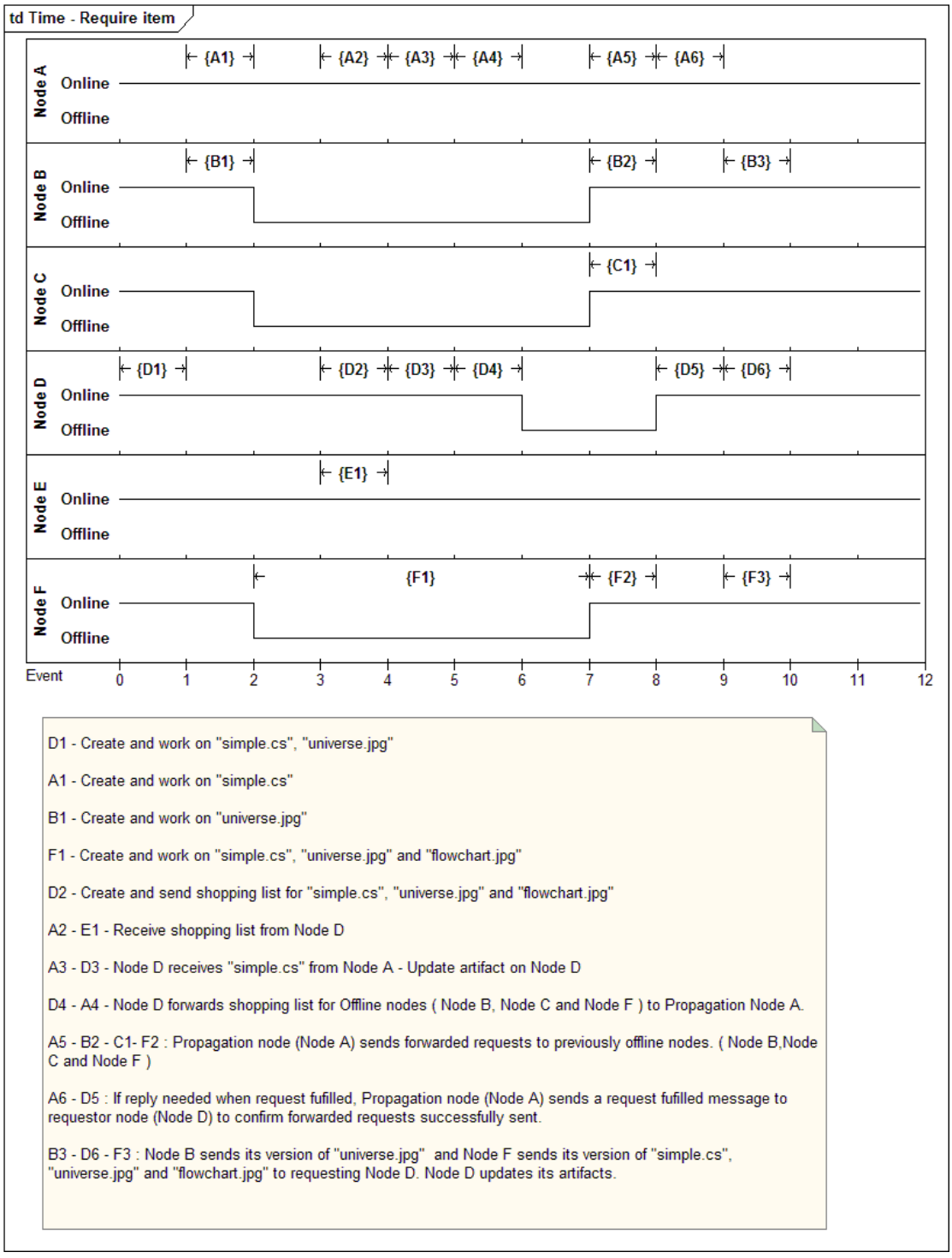


Figure 55 : Require item scenario timeline

Node D soon goes online and sends an online notification to all online nodes. At this stage, *Node B* and *Node F* respond to the shopping list and transfer their more updated versions of the artifact to *Node D*.

The forwarded requests are of type `FORWARDED_REQUEST_RESPOND_IF_REQUEST_FULFILLED` in this scenario. The propagation node, *Node A*, sends the requesting node, *Node D*, a *successful request fulfilled* response.

We note that since both *Node B* and *Node F* have updated versions of *"universe.jpg"*, artifacts from both nodes are transferred to the requesting node, *Node D*. If the user interaction is enabled, the user has the option of choosing which artifact they would rather prefer. An advantage of the artifact details shopping list, is that if *Node F* comes online before *Node B*, the more recent artifact would be transferred to *Node D* and updated. When *Node B* goes online, it would send the details pertaining to it's version of the artifact, and in response, *Node D* would not transfer the item as it is an older version. This is an issue related to timing and chance of the users in a real life scenario.

Table 9 shows the updated artifact list on *Node D* at the end of the scenario. We note that the *last author update* and *last artifact location* attributes of *"simple.cs"*, *"universe.jpg"* and *"flowchart.jpg"* denote that these artifacts have been updated or received from *Node F*.

Artifact Name	*simple.cs*	*universe.jpg*	*flowchart.jpg*
Size	170	1020	2020
Created	2/14/2006 11:39:18 PM703	2/14/2006 11:39:18 PM703	2/14/2006 11:39:27 PM46
Last update	2/14/2006 11:39:24 PM437	2/14/2006 11:39:27 PM31	2/14/2006 11:39:29 PM546
Author	D	D	F
Origin node	4	4	6
Last Author Update	F	F	F
Last Artifact Location	6	6	6

Table 9 : Final artifact list of Node D at end of scenario

Node D now has the latest versions of the requested artifacts that exist within the community. But what expense has it had on the other nodes, particularly the

propagation node, *Node A*? Figure 56 shows the total amount of data sent and received during this scenario where two shopping lists are sent.

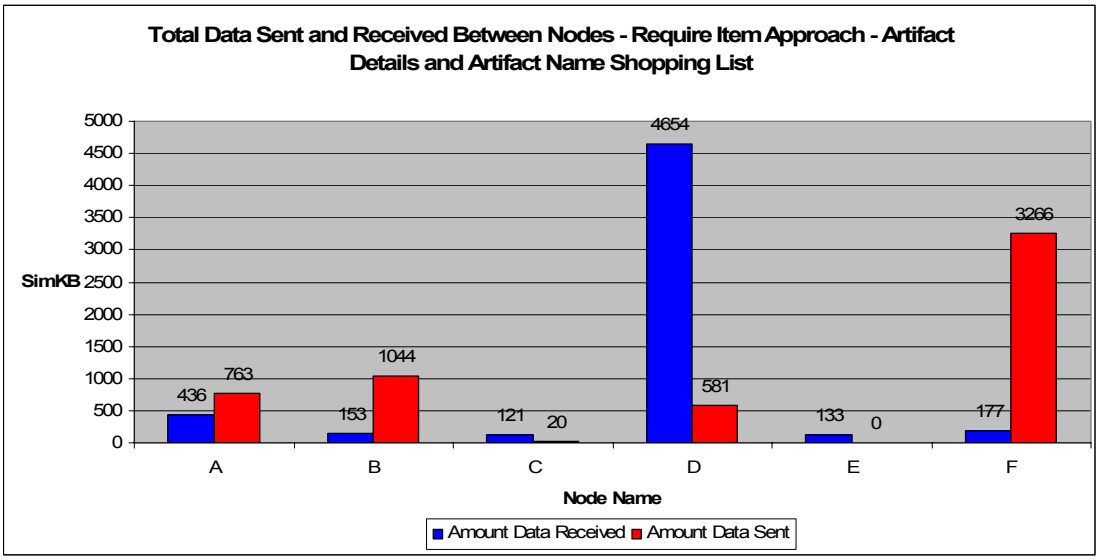


Figure 56 : Data transferred and received - Require item

The first forwarded shopping list is *64 simKB* and the second forwarded shopping list is *44 simKB*. Since three nodes were offline at the time of sending each of the two shopping lists, the data received on the propagation node is *((64 simKB * 3nodes) + (44 simKB * 3nodes)) = 324 simKB*.

The forwarded item is then *unwrapped* and transferred to the relevant nodes when they become available. The first unwrapped shopping list is *44 simKB* and the second is *24 simKB*. The amount of data transferred from the propagation node on behalf of the requestor node is *((44 simKB * 3nodes) + (24 simKB * 3nodes)) = 204 simKB*.

For each forwarded item, a *request fulfilled* response is sent. This response is 24 simKB and there were 6 forwarded requests made, hence an extra *(24simKB * 6 requests) = 144 simKB* that was transferred from the propagation node.

A total overhead of *324 simKB* is received and *348 simKB* is transferred on the propagation node. These are substantial amounts of data for in terms of the NPS.

Figure 57 shows the counters of each node in the simulation at the end of the scenario. *Node D* has attempted and failed to transfer the shopping item to nodes. The total of 30 retries is made of the five retries per shopping list for the three nodes.

The 12 *retry fails* are due to attempting and failing twice to send two shopping lists to three offline nodes and after the failed second attempt, we send forward the request to the propagation node.

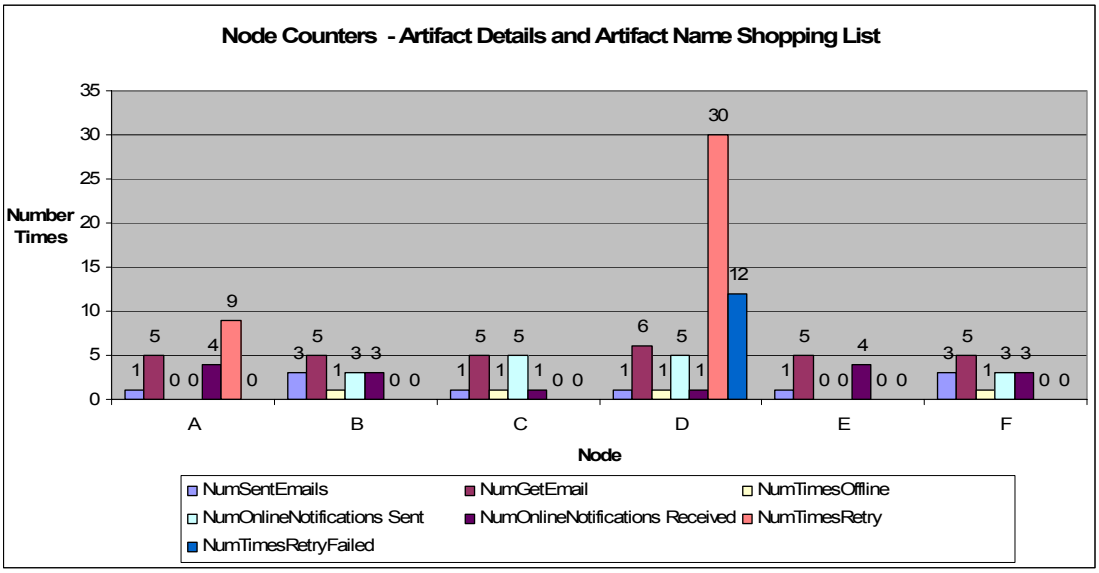


Figure 57 : Node Counters for require item approach

We note that once the propagation node receives the forwarded requests, it attempts to transfer the item in case its requestor node might not have the correct IP address of the assumed offline nodes. It fails the initial transfer but transfers the forwarded items once the nodes send online notifications. This can be seen in Figure 58.

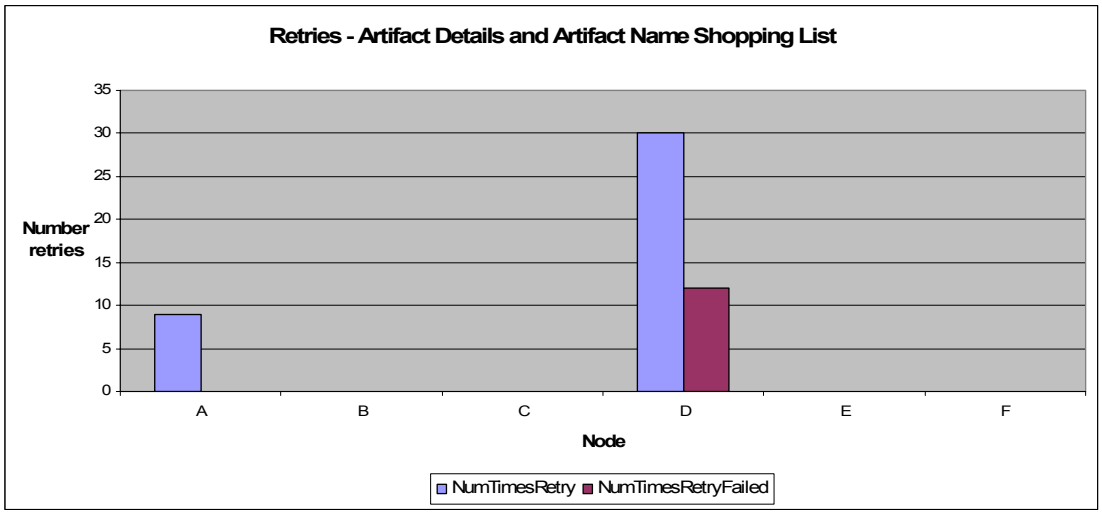


Figure 58 : Retry counts per node for require item approach

As mentioned in the previous case in this scenario (section 4.3.2), we could have opted for a single shopping list which contained all three artifact names. Figure 59 illustrates the data transferred and received between nodes with a single shopping list.

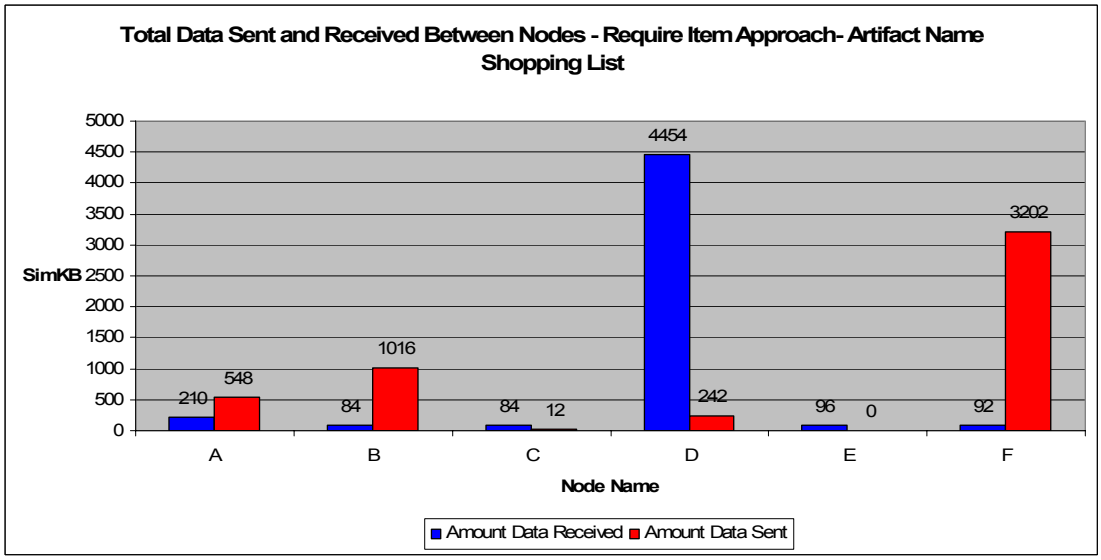


Figure 59 : Data transferred and received - require item approach with single shopping list

There is, overall, a slightly less amount of data being transferred and received when compared to Figure 56. This case would be a more viable option since the artifacts that are transferred are newer versions. The disadvantage would be if the transferred artifacts were older or the same version as that on the requesting node. If this were the case, we would have transferred a large amount of data which would ultimately be discarded by the requesting node.

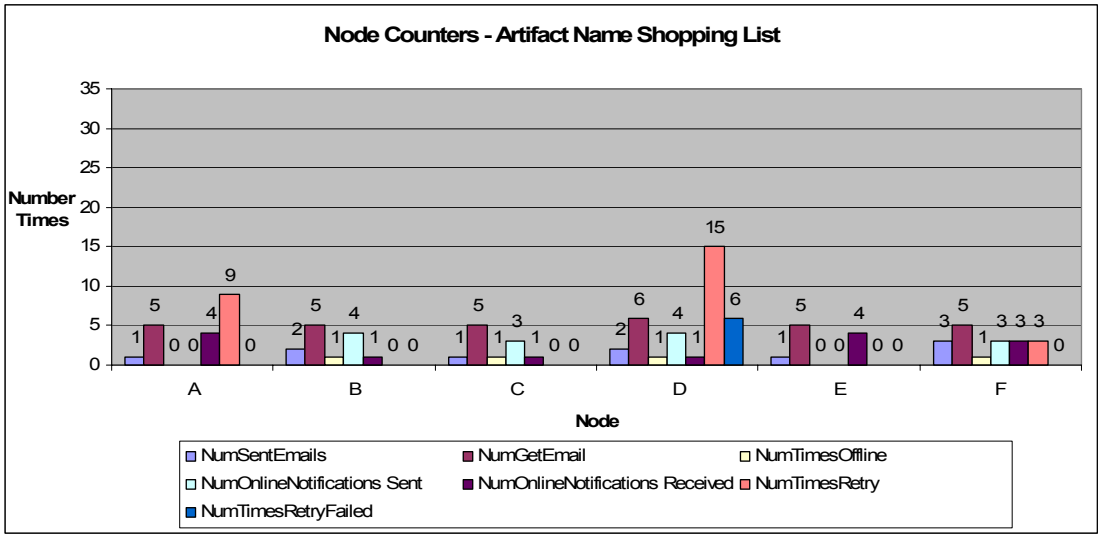


Figure 60 : Node Counters - require item approach with single shopping list

Figure 60 shows that the retry count for *Node D* has halved when a single shopping list is sent. In this particular instance, *Node F* attempts to transfer each of the updates before *Node D* if online again. This is not a cause for concern, but merely the timing between the threads in the NPS.

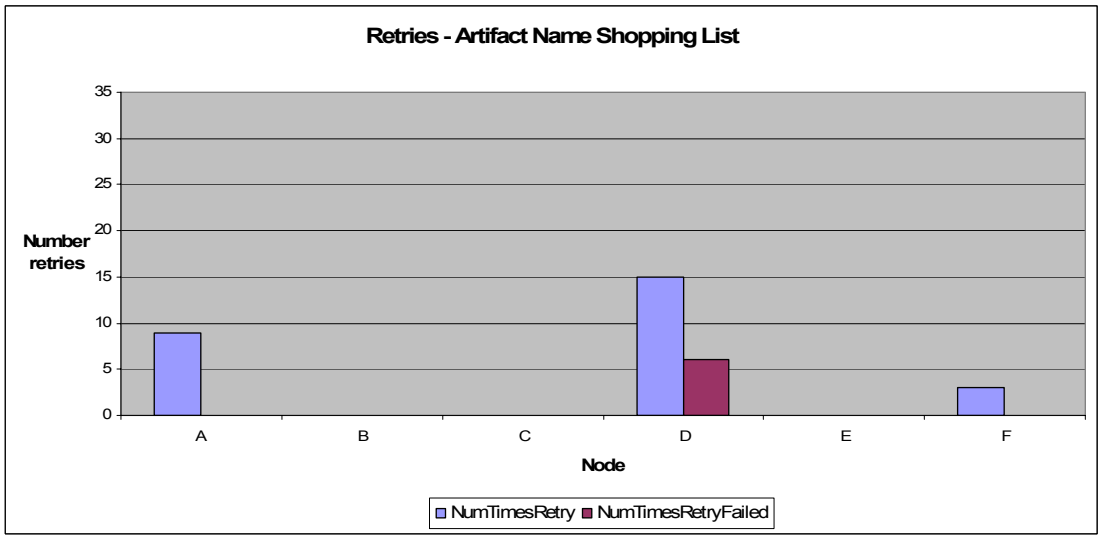


Figure 61 : Retries per node in require item approach with a single shopping list

Figure 61 shows the retry counts for each node. Although *Node F* fails on the first attempt to send the artifacts in response to the shopping list, on the second retry the artifacts are successfully transferred and the artifacts are updated on the requesting node, *Node D*.

4.3.4. Require item under any condition

In the previous cases, a request is discarded if it is not successful after a number of retries. The option exists to reset the *retry counter* for the item and continue to try to transfer the item regardless of the number of times the item has failed.

This option could be used in situations where a node has a weak signal and the connection is always disrupted. The disruptions caused the retry count to reach its maximum and the request can never be fulfilled.

This option would seem to have more disadvantages then advantages. We note some lessons that were learnt by using the NPS with this option. We could suffer from unnecessary data transfer overheads when the request becomes old. The request itself could become stale and the requestor might not need the request to be fulfilled after a certain number of attempts. Another disadvantage is that it becomes an overhead on the propagation nodes and new requests could suffer being satisfied. This is due to the fact that this request is never removed from the queue until it is completed. With user interaction however, the option exists to have this item discarded.

For these reasons, this option would not bear enough weight to be included in the Nomad protocol. If this option is to be implemented, it should be used with caution.

4.3.5. Conclusion

This section takes a look at the various effort levels that Nomad can offer its collaborators to gather artifacts in the community. The best effort approach, which will be the most used approach by collaborators, offers a simplistic approach to gather all artifacts from all nodes that are online at the time the shopping list is sent from the requestor node.

We introduced the use of a *propagation node*. Although this scenario makes use of a single propagation node, there can be more than one propagation node in a community. The transport item is forwarded to the propagation node after the requesting node has failed on a number of retries to send the item to other nodes. The propagation node hosts the transport item of the requestor until such time that the

nodes that were previously offline or unavailable, go online and become available to the community. This however creates an overhead on the propagation node in sending and receiving data on behalf of the requestor. Although a shopping list has been used in this scenario, the propagation node can host any transport item. We introduce the *optional* use of a *request fulfilled* response. This response is sent by the propagation node, denoting the success or failure of transporting the forwarded request.

The *require item under any condition* is an approach that bears more disadvantages than advantages. This approach could be used if the node has a very weak internet connection that is regularly disrupted and most likely would reach the maximum retry count easily. The item or forwarded item could however become stale. The node could suffer from the overhead caused in continuously retrying and failing. The item is kept in the queue and only removed if the transfer is successful. This option therefore could have dire consequences and should be used with caution.

The NPS shows that the *best effort* and *require item* approaches are sufficient for gathering artifacts from collaborator nodes.

5. Nomad and Mobile Agents

During the initial design reviews of Nomad, the use of mobile agents formed a large part the design. ([20],[31]) In this chapter, we take a broad look at mobile agents, their use and their intended integration into the Nomad application. The NPS does not offer mobile agents support, but is intended for future work.

5.1. Introduction

The concept of mobile agents is not novel. Mobile agents use the simple idea of moving the processing to the data, instead of the data to the processing. Mobile agents have been used in industry for many reasons and many situations. Example applications of mobile agents are electronic commerce, personal assistance, secure broking, distributed information retrieval, telecommunication network services, monitoring and notification, workflow applications and groupware, and parallel processing. [11]

So what is a mobile agent exactly? For the purposes of this text, we use the definition offered by Lange [11]:

> *'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.'*

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. The variety of mobile agent systems spans from, but not exclusive of, the operating system implementation, security, interoperability, mobility and the communication aspects. Lall [26] compares a variety of systems based on their security, mobility and the communication aspects and prepared a detailed analysis of a significant number of systems.

A few examples of Java based mobile agent systems are Aglets [11], NOMADS [28], JADE [43] and µCode [40]. There have been some initiatives to produce mobile agent systems using the Microsoft .NET Framework, namely MAPNET [12] and EtherYatri.NET [41]. Some systems, such as CARLA [27] and the Wireless Agent Simulator [34], have also explored the use of wireless mobile agents in handheld devices.

There are many aspects of mobile agents which lie outside the context of this text. Interested readers are directed to Lange [11], Rama [20] and Lall [26] for a more detailed overview of agents.

5.2. Motivation

For the first implementation of Nomad, shopping lists will form simple requests. Mobile agents would be useful when we have larger and more complex shopping lists made up of a variety of artifact details, where each artifact may have dependencies and the dependencies in turn could have dependencies. A user might opt to send a request out to the community and pick up the results at a later stage.

An example situation is that a collaborator, Himal, is at the airport waiting to board a flight. He realizes that he needs an updated artifact from the community. He invokes Nomad, which in turn, sends out an agent on his behalf. Himal then goes offline and boards his flight. While he is in flight, the agent moves from node to node looking for the artifact in question. In doing so, realizes that the artifact has dependencies and adds these artifacts to the shopping list. When Himal reaches his destination, he makes himself available to the community. The agent realizes that Himal has returned to an online status and returns to the mobile agent system on his device. The agent then updates Nomad with the needed information. The information contains, among other things, the users that the agent has visited, the users that have the needed artifact, and the dependencies of the artifact, and the users that have the dependencies of the artifact. At this point, the information is transparent to Himal. Nomad *intelligently* decides which node to download the artifact and dependencies from. This is dependent on a number of reasons, possibly transfer speed or availability of a node.

If Nomad cannot decide, it would invoke the user's interaction. Nomad would then download the required artifacts and update the project accordingly.

Mobile agents work well in casually connected environments. Lange [11] states that there are seven good reasons to use mobile agents. We state each of these points, explain them and relate them to Nomad.

1) *They reduce network load.*
   Distributed systems often rely on communication protocols that involve multiple interactions to complete a given task. This is especially true when security measures are active. When processing has to take place on large volumes of data, move the computations to the data rather then data to computations.
2) *They overcome network latency.*
   Critical real-time systems need to respond real time to changes in the environment. Nomad is not a time critical system, and does need to respond in real time.
3) *They encapsulate protocols.*
   A situation can arise when the hosts maintain their respective code that control outgoing messages. Protocols evolve to improve security and efficiency. Due to the new requirements, it then becomes cumbersome, if not impossible to upgrade the protocol code properly. This leads to the protocols becoming legacy code. Mobile agents move to the remote hosts to establish "channels" based on propriety protocols.
4) *They execute asynchronously and autonomously.*
   Mobile devices rely on expensive or fragile network connections. Tasks that require a continuous open connection between a mobile device and a fixed network will not be feasible economically or technically. Tasks can be embedded into mobile agents then dispatched into the network. After being dispatched, agents become independent of the creating process and can operate *asynchronously* and *autonomously*. The mobile device can reconnect at a later time to the network to collect the agent and hence the results.
5) *They adapt dynamically.*
   Mobile agents have the ability to sense their environment and react autonomously to changes.

6) *They are naturally heterogeneous.*
   Mobile agents are only dependent on their execution environment and are computer and transport layer independent. This provides seamless system integration.
7) *They are robust and fault tolerant.*
   The ability to react dynamically to unfavorable situations and events makes it easier to build robust and fault tolerant distributed systems.

By reducing network load, overall overhead on individual nodes is reduced. As seen in the NPS simulations earlier in this thesis, data received and transferred between nodes can bear much overhead on nodes. This is especially seen in the case of propagation nodes. Propagation nodes act on behalf of other nodes and therefore the data received and transferred for a forwarded request is overhead for the propagation node. Furthermore, an agent can move to *all* nodes in the community and decide that an artifact needs to be fetched from a single node. This reduces the amount of multiple artifacts that arrive at the requesting node.

Since agents have the ability to act autonomously and asynchronously, a user can opt to forward it to the propagation node. It will be able to detect that it is on a propagation node, and react dynamically accordingly to the environment, in this case, without user intervention since the user is offline. In the case of devices that have expensive internet connections, an agent will dynamically opt to send less data if possible. It would be possible to have less extensive processing, in terms of artifact comparisons, done on devices with small memory footprints.

The fact that they are heterogeneous, robust and fault tolerant, allows the agents to make decisions based on their environments that they are executing on. If the computer is about to shut down, the agent will be aware of having to 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* in Nomad would allow the agent to be independent of underlying protocols. Hence, in a way encapsulating the needed protocol used to move between systems.

5.3. Integration of mobile agents into Nomad

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. Rama [20] takes a more detailed look at this integration.

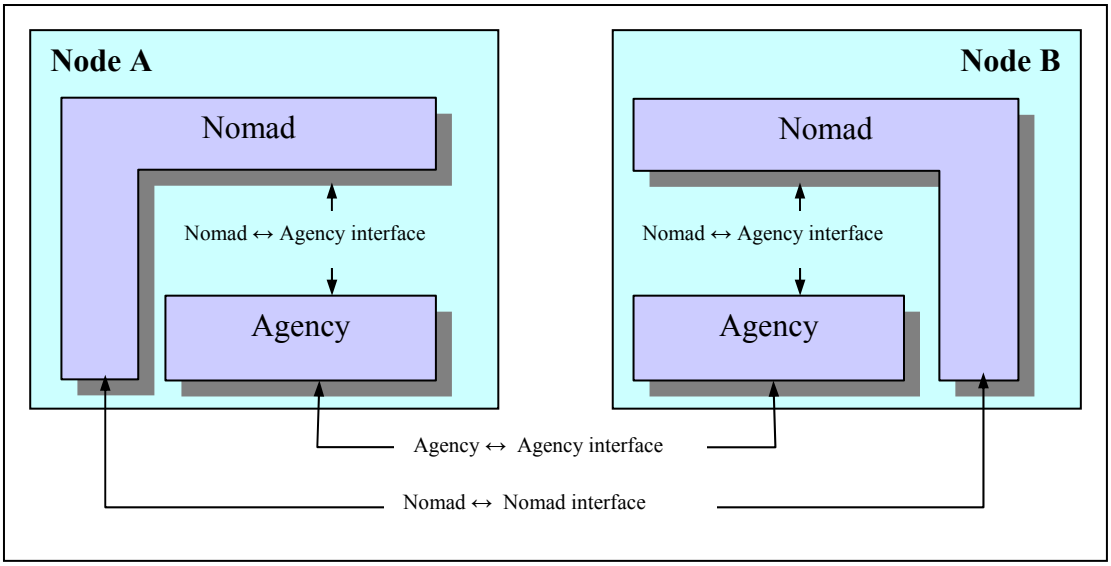


Figure 62 : Pluggable mobile agent system

With the initial design of Nomad, mobile agents would be a user option. The user could opt to use mobile agents. For this reason, it was decided that the mobile agent system be a *pluggable* system for Nomad. Figure 62 shows a high level illustration of the pluggable mobile agent system between two nodes. The Nomad communication channel will remain available for Nomad interactions. These interactions could be retrieving of artifacts or sending of online notifications. By no means is the intention of the mobile agent system to replace the Nomad system, but merely give it more flexibility and customizability.

The agency would relay information gathered from agents to Nomad. Nomad in turn would make the final decision. The agency would be transparent to user and all interactions from the agency to the user will be filtered through the Nomad system.

5.4. Final word and conclusion

Mobile agents have much value to add to Nomad. We have introduced the mobile agent, and given motivations as to why mobile agents would be useful to Nomad. We have given situations of how mobile agents could be used in Nomad and the idea of the agency being a pluggable system to Nomad.

Due to the overall complexity and overhead of a mobile agent system, mobile agents will not be included into the first release of Nomad. It was realized that the addition of a mobile agent system to the initial release of Nomad would not introduce any significant immediate gain to Nomad. Shopping lists are currently small and simple. The communities are small. Intelligent processing of requests would reduce initial overhead on nodes for the initial release. The limited number of nodes used for the first release would ultimately result in resource overhead. Once the amount of nodes in the Nomad exceeds the point where the overhead in the system adversely affects the performance, the use of agents would be influential. Mobile agents would be an optional add-on feature later in the lifecycle of Nomad. We intend to introduce performance criterion as to whether agents in fact do add significant gain to Nomad.

6. Evaluation

6.1. Other Nomad Related Projects

During the requirements phase of Nomad in 2004, comparisons to other systems were done to evaluate Nomad. We evaluated three commercial and three academic systems. The following were the results of our findings [21]. We would like to thank the Nomad members for their effort in this process.

The commercial products are:

- Novell iFolder [48]
- SubethaEdit [50]
- Microsoft Sharepoint [44]

The academic systems are:

- CoCoDoc [13]
- Basic Support for Collaborative Work (BSCW) ([4],[5],[6],[7])
- X-peers [52]

We give a short introduction to each of these systems. Each system is described by extracts from related references given above. The reader is asked to follow references given for each system for more detailed information.

6.1.1. Systems overview

6.1.1.1. Novell iFolder

> *"Novell iFolder®, which ships in Novell® Open Enterprise Server, allows your files to automatically follow you everywhere-online and offline-across multiple systems and the Internet. Any changes you make to a Novell iFolder directory are automatically and intelligently updated to your company's Novell iFolder server and your other computers through your Internet connection. You can also share your files with others in the network."*

6.1.1.2. SubethaEdit

"SubEthaEdit is a powerful and lean text editor. And it's the only collaborative one you can actually use. By combining the ease of Bonjour with the world's best text collaboration engine, it makes working together not only possible but even fun [..]Editing documents in groups can be a challenge. Versioning systems like subversion or cvs help your group to keep a consistent copy of your document, but don't provide realtime collaboration. Wouldn't it be great to edit the same document, live, in realtime, together with everyone in your group?"

6.1.1.3. Microsoft Sharepoint

"SharePoint Products and Technologies facilitate collaboration within an organization and with partners and customers. Using the combined collaboration features of Microsoft Windows SharePoint Services and Microsoft Office SharePoint Portal Server 2003, users in your organization can easily create, manage, and build their own collaborative Web sites and make them available throughout the organization."

6.1.1.4. CoCoDoc

"We propose collaborative compound document editing as a new paradigm for editing environments and describe the design and implementation of CoCoDoc, a framework based on OpenDoc and CORBA. CoCoDoc supports reuse of existing editors as simple collaborative editors and supports development of new collaborative compound part editors with flexible collaboration facilities, thus facilitating a gradual migration towards collaborative editing environments that are both rich in editing support and rich in collaboration support."

6.1.1.5. Basic Support for Collaborative Work (BSCW)

"The BSCW system supports collaboration by providing shared workspaces over the Internet. A shared workspace allows storage and retrieval of documents and sharing information within a group. This functionality is integrated with an event mechanism to provide each user with an awareness of the activities of others within the workspace. It comprises numerous features,

e.g., support for threaded discussions, version management of documents, group management, search features and many more. The system is designed primarily to support self-organising groups."

6.1.1.6. X-peers

"xpeers is for group communications. It is designed to help people work together in a secure fashion from anywhere and with almost any type of computer system. The unique power enables an unlimited range of new applications for information sharing, collaboration, and coordination. The basic installation of xpeers solves three core-communication problems:

- *Information sharing: xpeers keeps every member in a group in sync. Each member has a copy of the same version of a file for offline access. xpeers keeps track what and when to sync. You can add every filetype to the xpeers universe.*
- *Communication: xpeers provide private and public instant messaging. You can create as many topic groups as you like.*
- *Versioning: xpeers has a built in version control system. You only have one file on your system, the most recent one. You don't need to keep older version yourself, the xpeers server has all versions. You can recall a version at any time."*

6.1.2. Systems comparison

We compare each system based on criteria which are common in Groupware applications.

6.1.2.1. Functional Criteria

Groupware applications can be described by its functional criteria. The functional criteria specify what functionalities a user can expect of the system regardless of its environmental or non-functional constraints.

- **Messaging**

Provide users the functionality to communicate via synchronous and asynchronous messages. Synchronous messaging systems are Instant Messaging

systems (IRC, MSM) while asynchronous systems are mostly email-type applications.

- **Conferencing and Electronic Meeting Systems (EMS)**

Conferencing and electronic meeting systems provide users with a shared communication channel, interface or workspace where they can work, talk or share simultaneously.

- **Group Decision Support**

Group decision support systems are portal-like applications that ensure all collaborating users have access to the same, accurate and most up to date data on a given subject.

- **Document Management**

Provide document management features such as indexing, searching and distributing documents to authorized users.

- **Document Collaboration**

Document collaboration systems include document management functionalities and extend them by providing history, versioning and change management.

- **Compound Document Management**

Provide the functionality to see one document as a collection or combination of smaller documents. Compound document management systems allow users to view the single merged or compound version of the document.

	Messaging	Conferencing & EMS	Group Decision Support	Document Management	Document Collaboration	Compound Document Management
Nomad				☑	☑	☑
Novell iFolder				☑		
SubethaEdit	☑	☑		☑	☑	
Microsoft Sharepoint			☑			
CoCoDoc				☑	☑	☑
Basic Support for Collaborative Work (BSCW)	☑	☑		☑	☑	
X-peers	☑			☑	☑	

Table 10 : Functional criteria

6.1.2.2. Architectural Criteria

The architectural criteria of Groupware applications define where and how the collaboration is managed.

- **Central Architecture**

Collaboration is managed at a central server. All data is exchanged via a central point of access. This architecture relates directly to client-server architecture.

- **Replicated Architectures**

Collaboration is managed by all the peers in the network. Data and information are exchanged between peers and all peers are equal in such a network. This architecture relates to pure Peer-to-Peer architectures.

- **Hybrid Architecture**

The Hybrid Architecture can be seen as a Peer-To-Peer architecture where some nodes are more significant than other nodes. Data and information are shared among the peers, but there exists a single or multiple master peers that can override information received from peers or they can guide peers to other nodes. This architecture relates to Peer-to-Peer networks with *super-nodes*.

	Central	Replicated	Hybrid
Nomad			☑
Novell iFolder		☑	
SubethaEdit		☑	
Microsoft Sharepoint	☑		
CoCoDoc	☑		
Basic Support for Collaborative Work (BSCW)	☑		
X-peers	☑		

Table 11 : Architectural Criteria

6.1.2.3. Focus Criteria

Focus area criteria defines the focal point of collaboration. This means that all collaborative tasks are centered on this particular area and that no collaboration is possible without the focus criteria being present.

- **User Centered**

Collaborative tasks focus on the user. This implies that the user is the most important aspect in user centered collaboration. User centered groupware creates a communication channel between collaborating users; the groupware is not interested in what the users do with the channel.

- **Artifact Centered**

All collaborative tasks focus on the artifact. Artifact centered Groupware provide methods to collaborate on a specific artifact. The Groupware will typically store information with regards to the structure and history of the artifact. The communication channel between users is transparent.

- **Workspace Centered**

Workspace centered Groupware can be seen as an extension to user-centered Groupware with the exception that a workspace can exist without users. The *workspace* can store the state and in this way allows asynchronous user centered collaboration. Collaborative users share the same workspace.

	User Centered	Artifact Centered	Workspace Centered
Nomad		☑	
Novell iFolder		☑	
SubethaEdit	☑		☑
Microsoft Sharepoint		☑	
CoCoDoc			☑
Basic Support for Collaborative Work (BSCW)	☑		☑
X-peers			☑

Table 12 : Focus Criteria

6.1.2.4. Time Criteria

Time criteria define the restrictions placed on the time of collaboration.

- **Synchronized**

Collaboration must happen in a structured manner at the same time. Synchronized Groupware will handle locking and collision detection in real-time.

- **Unsynchronized**

Collaboration can happen entirely unsynchronized. Unsynchronized

Groupware supports people working together, completely separate from each other. Collaboration only kicks in when requested from a user, otherwise all work performed does not affect other collaborating users.

- **Mixed (Synchronized & Unsynchronized)**

Collaboration can be either synchronized or unsynchronized.

- **Serial**

Serial collaboration is unsynchronized with the exception that one user must perform a specific task before another user can continue with another task. Email is a classical example of serial collaboration.

	Synchronized	Mixed	Serial	Unsynchronized
Nomad				☑
Novell iFolder				☑
SubethaEdit	☑			
Microsoft Sharepoint			☑	
CoCoDoc		☑		
Basic Support for Collaborative Work (BSCW)		☑		
X-peers		☑		

Table 13 : Time Criteria

6.1.2.5. Platform Criteria

The platform criteria define the execution platform for the groupware application.

- **Mobile Platforms**

Collaboration can be extended onto mobile and handheld devices.

- **Operating System based Platform**

Collaboration can only occur on nodes sharing the same operating system.

- **Browser based platforms**

Collaboration can occur via any Web browser.

- **Platform independent (Multi-platform)**

Collaboration can occur on multiple platforms. These solutions are either built on top of runtimes such as Java or .NET or there exist a binary distributable version for most platforms.

	Mobile Platforms	Operating System Based Platforms	Browser Based Platforms	Platform Independent
Nomad	☑			☑
Novell iFolder	☑		☑	☑
SubethaEdit		☑	☑	
Microsoft Sharepoint	☑	☑		
CoCoDoc		☑		
Basic Support for Collaborative Work (BSCW)			☑	
X-peers				☑

Table 14 : Platform Criteria

At the time of writing:

- Nomad uses .Net and .Net compact framework.
- Novell iFolder runs on Novell Netware.
- X-Peers have binary versions available for various operating systems.
- SubethaEdit runs Mac OS X only.

6.1.2.6. User involvement Criteria

Defines the level of involvement required from the user to gain advantages provided by the groupware.

- **High**

  High user involvement means that the user is forced to work with a different interface that he is used to in order to access the collaboration functionalities. This is typical to shared workspace environments.

- **Medium**

  Medium user involvement implies that users can work with their normal user interfaces and only need to execute collaborative commands at any given time.

- **Low**

  Low user involvement means that the user is only involved in setting up the collaboration environment and can then continue to work as if they are not collaborating. All collaboration functions are automated and intended to be transparent to the user.

	Low	Medium	High
Nomad	☑		
Novell iFolder	☑		
SubethaEdit			☑
Microsoft Sharepoint			☑
CoCoDoc			☑
Basic Support for Collaborative Work (BSCW)			☑
X-peers		☑	

Table 15 : User Involvement Criteria

6.1.3. Discussion

Table 16 shows the combination of the systems against all criteria. We note that there exist commonalities between the systems. All of the above systems:

- Work with closed communities or groups. There is no public file sharing. All users are aware of all other collaborators in the community.
- Focus is on *collaboration*; not sharing. This means they work with dynamic documents (documents that change over time) rather than sharing which focuses on static documents.
- Execute in a distributed environment using common network protocols such as TCP and HTTP, with the internet as backbone.

On a functional level, the *CoCoDoc* framework is the closest match to Nomad. Both provide a framework to support collaboration on compound documents. Nomad extends this by adding Group Document Management features.

Microsoft Sharepoint is the only "group decision support" and "serial" project. This product is least similar to Nomad.

Criteria			System						
			Nomad	Novell iFolder	SubethaEdit	Microsoft Sharepoint	CoCoDoc	BSCW	X-peers
Criteria	Functional	Messaging			☑			☑	☑
		Conferencing & EMS			☑			☑	
		Group Decision Support				☑			
		Document Management	☑	☑	☑		☑	☑	☑
		Document Collaboration	☑		☑		☑	☑	☑
		Compound Document Management	☑				☑		
	Architectural	Central				☑	☑	☑	☑
		Replicated		☑	☑				
		Hybrid	☑						
	Focus	User Centered			☑			☑	
		Artifact Centered	☑	☑		☑			
		Workspace Centered			☑		☑	☑	☑
	Time	Synchronized			☑				
		Mixed					☑	☑	☑
		Serial				☑			
		Unsynchronized	☑	☑					
	Platform	Mobile Platforms	☑	☑		☑			
		Operating System Based Platforms			☑	☑	☑		
		Browser Based Platforms		☑	☑			☑	
		Platform Independent	☑	☑					☑
	User	High			☑	☑	☑	☑	
		Medium							☑
		Low	☑	☑					

Table 16 : Combined systems comparison

Other criteria exist, such as quality of service and security, which were not included. They are not relevant to the onset of this thesis but might be worthwhile to research.

6.2. Comparison of similar mobile agent 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.

Name	Description	Language / Architecture	Wireless Connectivity	Other
Aglets [11]	API from IBM used to program mobile internet agents	Java	No	Free, Open source, Educational.
CARLA [27]	CORBA based Lightweight agents.	Java and CORBA	Yes	Aimed at providing architecture for handheld devices.
µCode [40]	Mobile code toolkit	Original in Java with .NET ported version available.	No	Open source, Educational.
SWAN [34]	SWAN provides a test-bed for devices that make use of ad hoc networks.	Java	Yes	Uses JADE [43] as underlying Agent framework.
MAPNET [12]	Mobile Agent Framework	C# and .NET	No	Follows the MASIF specification, Educational.
NOMADS [28]	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 [45]	Research project aimed to support nomadic users in the near future.	Java	Yes	Most closely related to Nomad.

EtherYatri.NET [41]	Mobile agent toolkit	C# and .NET	No	Free, Open source, Educational, integrates into Visual Studio
Nomad [20]	Aims to support nomadic users. Use of mobile agents to add functionality.	C#, .NET and the .NET Compact Framework	Yes	Educational, provides collaboration between users with variety of devices.

Table 17 : Nomad in comparison with other mobile agent projects

Aglets are 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 in the literature we encountered. 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 minimumCORBA 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.

Simulator for Wireless Ad-hoc Networks (SWAN) makes use of the *Java Agent Development Framework* (JADE). SWAN along with the *Wireless Agent Simulator* (WAS) was written in Java and was implemented on top of the FIPA-based JADE [34]. 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 make 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 to our knowledge has functionality for 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 of 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 the time of writing 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.

6.3. Enabling technologies

This section mentions a few of the many features that have a direct impact on how designers develop distributed systems and particularly features that will be used in Nomad. 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.

6.3.1. .NET Framework

The .NET Framework is a platform that aims to simplify application development in the highly distributed environment of the Internet. As opposed to the virtual machine in Java, the .NET version of a virtual machine is the Common Language Runtime (CLR). The topic of the CLR is not within the scope of this thesis, though it is necessary to mention it, as it serves as the backbone to .NET. Some of the features as noted by Powell [30] 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 allows for multiple language support.* 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 Intermediate Language (IL). 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#.

6.3.1.1. 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.

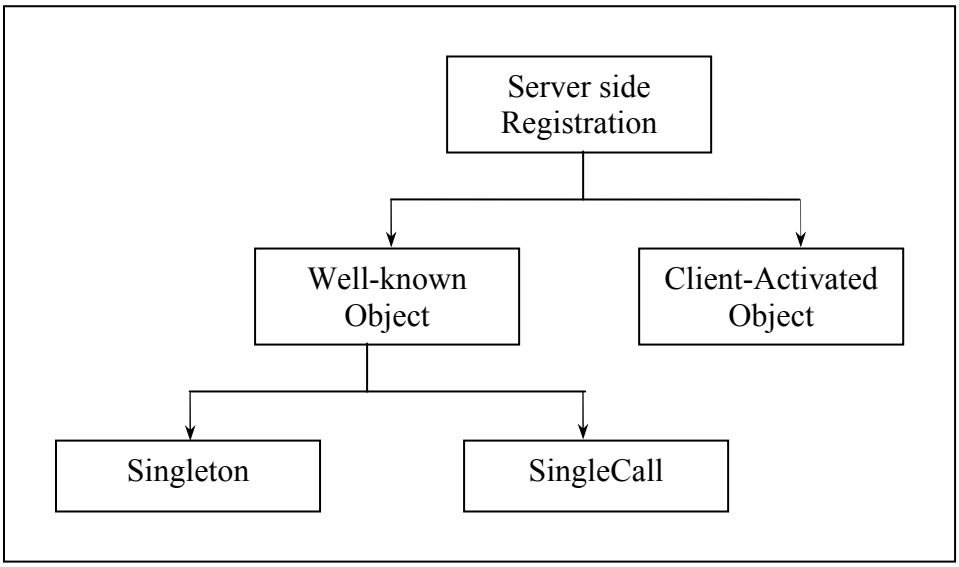


Figure 63 : Remoting Server Object Types

There are two types of server object types that .NET supports: *well-known objects* and *client-activated objects*, as shown in Figure 63. 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, who 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 *transport items* in Nomad and for transferring of 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.

6.3.1.2. Reflection - System.Reflection

In section 5 of this text, we discuss the possible use of mobile agents in Nomad. 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 [46]. From the assembly, the identity of the agent 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.

6.3.1.3. .Net Version 2.0

The NPS was written in .Net Version 1.1. At the time of writing, and during the development of Nomad, version 2.0 of the .Net framework was released. Although the NPS was already developed, it was then realized that .Net version 2.0 had much to offer Nomad. We note a few features of .Net version 2.0 as stated by MSDN [47] which would be of interest to Nomad.

Access Control Lists (ACL) is used to grant or revoke permission to access a resource on a computer. An ACL could be used to allow Nomad to view only certain sections of a computer, thereby allowing only that section to be seen by other nodes. This would help to aid in privacy of data on nodes.

The *NetworkChange* class allows applications to receive a notification when the status of the network connectivity of a node changes. Nomad could use this feature to automatically send requests when the network becomes available. Other network features of interest are the *Ping*, and *System.Net.NetworkInformation* classes. The *Ping* class determines if a remote node is available over the network. The *System.Net.NetworkInformation* class can access useful network traffic stats which might be useful in determining whether a large artifact could indeed be sent over the current connection, or should be queued for transfer at later time.

File Transfer Protocol (FTP) resources have been made available via the *System.Net* namespace. FTP would easily allow file transfer between nodes.

An interesting feature is seen in the *System.IO.Compression* namespace. This namespace offers applications to read and write data with the GZIP compression algorithm. This would greatly reduce the amount of data transferred between nodes.

The *System.Net.Mail* and *System.Net.Mime* namespaces allow email to be sent from an application. Since Nomad relies on email when nodes have become disconnected with new email addresses, these namespaces would certainly hold promise for Nomad.

Nomad relies on messages being sent between nodes. At the time of writing, it was decided that messages would be sent in *eXtended Markup Language* (XML) format. The new version of .Net offers a variety of features relating to XML which will be of interest to Nomad and offer an easier alternative to transfer messages.

The *System.Runtime.Remoting* and *Serialization* namespaces form an important aspect of Nomad. They have been updated in the newer version of .Net, but no immediate gain was foreseen in the newer versions for use in Nomad.

6.3.1.4. 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.'* [46]

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

Transport items would need to be serialized to be able to be transferred 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.'*[46]

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. Transport items would follow the same process.

6.3.2. .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 [35] 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.

6.3.2.1. Lightweight Nomad

It would have been the ideal if we were able to run the Nomad desktop version on devices such as a Pocket PC. Unfortunately, a lightweight version is needed due to the following reasons:

- Although the compact framework allows WiFi connections by simply making IP connections, at the time of writing, 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 window's 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 [27] has produced a similar lightweight concept in CORBA. The *CORBA-based Architecture for Lightweight Agents* or CARLA project focused on the *minimumCORBA* 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.

6.4. Conclusion

This chapter evaluates a number of CSCW groupware systems against criteria which we devised. The systems include three commercial systems and four academic systems, one which is our own system, Nomad. Furthermore, we evaluate Nomad with other mobile agent systems. We note the similarities and differences of these systems and Nomad in both evaluations. We conclude by discussing the enabling technologies, the .Net and .Net compact framework, and features thereof, which will allow us to make Nomad a reality. We state that future work will include a lightweight version of Nomad for use on devices with limited memory and processing power, yet host a variety of connectivity protocols.

7. Conclusion and future work

Nomad is a CSCW groupware tool which allows globally dispersed members in a virtual community to share information. The community is broken into project groups who share a common goal. The granularity of the information is dependent on the members of the community. We use artifacts to denote the piece of shared information. This could vary from, but not exclusive to, a picture, a sound clip, a paragraph or a chapter in a book. Nomad makes use of technologies such as peer to peer to make this a reality even though limited to a single depth search. This is a viable option since each member is aware of every other member in the project group. Members in the group switch from being online to offline at any time and for this reason, we cannot assume a constant connection to the community.

This thesis simulates an underlying protocol which allows Nomad members to collaborate. A variety of devices could be used in the community and we are therefore faced with a number of different protocols and bandwidth speeds characterized by each of these devices. A proof of concept, known as the Nomad Protocol Simulator (NPS), was developed to model and simulate the high level interaction of nodes based on events. The results communicated in this thesis are based on the finding from the NPS. We simulate three possible node types, although other types can be made available, and demonstrate how the protocol can be customized to adapt to these nodes based on their characteristics. We illustrate the use of email to set up the community and keep nodes in sync.

Artifact transfer poses a problem if there are a number of nodes that have the same or older version of a required artifact. Artifacts could be transferred to the requesting node regardless of the version. This might not pose a problem if the artifacts are small. But it the artifacts become rather large, bandwidth of both requestor and provider can be wasted. If the artifact exists on the requesting node, we could send the details as part of the request and therefore only send newer versions to the node. It is also possible to send the details of artifacts that reside on the providing node, and allow the requesting node to decide which version it wants.

The *best effort* approach will be the most used method of gathering artifacts. This method allows artifacts to be gathered from nodes that are currently online. Artifacts that exist on nodes that are offline, although possibly more updated, will not form part of this search. If, however, an artifact is required from a collaborator node that is never online at the same time as the requesting node, a propagation node is used via a *require item* approach. We evaluate the overhead of both approaches on both online and offline nodes. The use of a propagation node to host the request and the artifact of both nodes is also evaluated. We demonstrate that this solution, although viable, creates overhead on the propagation node. Therefore, even though any node in the group could be a propagation node, we define certain characteristics which the node would need to have before allowing it the status of a propagation node. Furthermore, we establish that the *require item under any condition* approach bears more disadvantages than advantages and if considered, should be used with caution. This is since the item or forwarded item could become stale and the nodes concerned could suffer from the overhead caused in continuously retrying and failing. We conclude that that the *best effort* and *require item* approaches are sufficient for gathering artifacts from collaborator nodes.

This thesis suggests the use of mobile agents in Nomad. We evaluate Nomad against other mobile agent systems and show their potential use in Nomad. Due to the limited number of nodes in Nomad and the simplicity of shopping lists, it was realized that the addition of a mobile agent system to the initial release of Nomad would not introduce any significant immediate gain to Nomad. Once the amount of nodes in the Nomad exceeds the point where the overhead in the system adversely affects the performance, the use of agents would be influential. Mobile agents would an optional add-on feature later in the lifecycle of Nomad. Since mobile agents pose potential security risks, for example rogue agents or agents who “steal” the identities of other agents. Future work intends to introduce performance criterion as to whether agents in fact do add significant gain to Nomad and the security risks that mobile agents could pose.

Nomad being a CSCW tool is evaluated against other CSCW tools. We see similarities and distinguishing factors of Nomad. Furthermore, we suggest enabling

technologies such as .Net and .Net compact frameworks and their use in developing Nomad. Future work includes the lightweight version of Nomad in the .Net compact framework.

Communities are intended to be small and secure. Every collaborator is aware of every other collaborator in the group. One collaborator might belong to more than one project group. We are therefore faced with shared resources between projects which could lead to a security risk. Further work would include exploring the security related aspects concerning shared resources within the community and security against outside intruders.

We conclude by stating that this thesis provides a possible viable solution, based on the findings of the NPS, for Nomad users to collaborate in casually connected environments, where users are mobile for most part of the project.

8. References

[1] A. Iamnitchi, M. Ripeanu, and I. Foster. *Small-world filesharing communities*, IEEE infocom, 2004.

[2] Adriana Iamnitchi, Matei Ripeanu, and Ian T. Foster. *Locating data in (small-world?) peer-to-peer scientific collaborations*, In Proc. of the First Intl. Workshop on Peer-to-Peer Systems, Cambridge, MA, USA, 2002.

[3] Ao Tang, Cedric Florens, and Steven H. Low. *An empirical study on the connectivity of ad hoc networks*, Aerospace Conference, IEEE, (3) 1333-1338, March 2003.

[4] Appelt, W, *WWW Based Collaboration with the BSCW System*, Proceedings of SOFSEM'99, Springer Lecture Notes in Computer Science 1725, Milovy (Czech Republic), 66-78, December 1999.

[5] Appelt, W.: *What Groupware Functionality do Users Really Use?,* Proceedings of the 9th Euromicro Workshop on PDP 2001, Mantua, IEEE Computer Society, February 2001.

[6] Bentley, R., Appelt, W., Busbach. U., Hinrichs, E., Kerr, D., Sikkel, S., Trevor, J. and Woetzel, G., *Basic Support for Cooperative Work on the World Wide Web*, International Journal of Human-Computer Studies: Special issue on Innovative Applications of the World Wide Web, Academic Press, **46**(6)827-846, June 1997.

[7] Bentley, R., Horstmann, T., Sikkel, K. and Trevor, J., *Supporting Collaborative Information Sharing with the World Wide Web: The BSCW Shared Workspace System*, in The World Wide Web Journal: Proceedings of the 4th International WWW Conference, Issue 1, 63-74, December 1995.

[8] Bonnie A. Nardi, Diane J. Schiano, Michelle Gumbrecht, *Blogging as social activity, or, would you let 900 million people read your diary?* , [8], 222-231 2004.

[9] CSCW 2004, Proceedings of the 2004 ACM conference on Computer supported cooperative work, 2004, ISBN:1-58113-810-5.

[10] Daniel B. Horn, Thomas A. Finholt, Jeremy P. Birnholtz, Dheeraj Motwani, and Swapnaa Jayaraman, *Six Degrees of Jonathan Grudin: A Social Network Analysis of the Evolution and Impact of CSCW Research*, [8], 582-591, 2004.

[11] Danny B. Lange and Mitsuru Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, Reading MA: Addison- Wesley, 1998.

[12] Dilyana Staneva, Denitsa Dobreva, *MAPNET: A .Net-Based Mobile-Agent Platform,* International Conference on Computer Systems and Technologies - CompSysTech' 2004

[13] G.H. ter Hofte and H.J. van der Lugt, *CoCoDoc : A framework for collaborative compound document editing based on OpenDoc and CORBA,* Proceedings of the IFIP/IEEE international conference on open distributed processing and distributed platforms, 15-33, May 1997.

[14] Gerald Reif, Engin Kirda, Harald Gall, Gian Pietro Picco, Gianpaolo Cugola, and Pascal Fenkam. *A Web-based peer-to-peer architecture for collaborative nomadic working*, In Proceedings of the 3rd International Workshop on Web-based Infrastructures and Coordination Architectures for Collaborative Enterprises, co-located with the 10th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001). MIT, Cambridge (MA, USA), June 2001.

[15] Gianpaolo Cugola and Gian Pietro Picco. *Peer-to-peer for Collaborative Applications*, In Proceedings of the International Workshop on Mobile Teamwork Support, co-located with the 22nd International Conference on Distributed Computing Systems, Vienna (Austria), H. Gall and G.P. Picco eds., IEEE press, 359-364, July 2002.

[16] Grudin, J. *CSCW: History and focus,* IEEE Computer, **27** (5), 19-26, 1994.

[17] J. Chu, K. Labonte, B. N. Levine. *Availability and locality measurements of peer-to-peer file systems*, In ITCom: Scalability and Traffic Control in IP Networks, Proceedings of SPIE, (4868) Jul. 2002.

[18] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham. *Consistency maintenance in peer-to-peer file sharing networks*, Proc. of WIAPP'03, 3rd IEEE Workshop on Internet Applications, San Jose, CA, 76-85, June 2003.

[19] Jeffrey D. Campbell, *Instant messages: a framework for reading between the lines*, [8], 519-522, 2004.

[20] Jiten Rama and Judith Bishop. *Towards a mobile agent framework for Nomad using .Net,* Technical report, Polelo Research Group, University of Pretoria, 2005.

[21] Judith Bishop, Theo Danzfuss, Ronald Klazar, Jiten Rama and Tebalo Tseoali, *Nomad - Collaborative groupware for casually-connected communities*, Technical report, Polelo Research Group, University of Pretoria, 2005.

[22] K. Cheverst and G. Smith. *Exploring the notion of information push and pull with respect to the user intention and disruption*, International workshop on Distributed and Disappearing User Interfaces in Ubiquitous Computing, 67-72, 2001.

[23] Liam J. Bannon , Kjeld Schmidt, *CSCW: Four Characters in Search of a Context*, In J. Bowers & S. Benford (Eds.) Studies in Computer Supported Cooperative Work: Theory, Practice and Design. Amsterdam North-Holland, 3-16, 1991.

[24] M.S. Khambatti, K.D. Ryu, and P. Dasgupta. *Push-pull gossiping for information sharing in peer-to-peer communities*, Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, Nevada, 1393-1399, June 2003.

[25] Manfred Hauswirth, Mehdi Jazayeri, *A Component and Communication Model for Push Systems*, ESEC / SIGSOFT FSE, 20-38,1999.

[26] Manoj Lall, *Selection of mobile agent systems based on mobility, communication and security aspects*, M.Sc dissertation, UNISA, 2005.

[27] Markus Aleksy, Axel Korthaus and Martin Schader, *Design considerations for a CORBA-based architecture for lightweight agents (CARLA)*, Web Intelligence and Agent Systems: An international journal1, 259-271, 2003.

[28] Niranjan Suri, Jeffrey M. Bradshaw, Maggie R. Breedy, et al, *An Overview of the NOMADS Mobile Agent System*, In Proceedings of ECOOP'2000, Nice, France, 2000.

[29] R. Baghwan, S. Savage, G, M. Voelker. *Understanding Availability*, Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Feb. 2003.

[30] Robert, Powell and Richard Weeks, *C# and the .NET Framework, The C++ Perspective*, SAMS, Indiana, USA, 2001.

[31] Ronald Klazar, Nomad, *Artefact Ubiquity, Extending the Reach of Collaborators by Means of Mobile Software Agents*, Technical report, Polelo Research Group, University of Pretoria, 2005.

[32] S. Saroiu, K. P. Gummadi, and S. D. Gribble. *Measuring and analyzing the characteristics of napster and gnutella hosts,* Multimedia Systems, (9)170-184, 2003.

[33] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. *A survey of peer-to-peer content distribution technologies*, ACM Computing Surveys, 36(4):335-371, December 2004.

[34] Stephen Quirolgico, L. Jay Wantz, et al. *Wireless Agents in Ad Hoc Networks*, Innovative Concepts for Agent-Based Systems, First International Workshop on Radical Agent, WRAC 2002, Springer, 165-174, 2002

[35] Steve Makofsky, *Pocket PC Network Programming*, Addison Wesley Professional, 2004.

[36] Tsoumakos, D. and Roussopoulos, N. *A comparison of peer-to-peer search methods,* In Proceedings of the Sixth International Workshop on the Web and Databases, San Diego, CA, 2003.

[37] Wikus Coetser and Judith Bishop, *A unified approach to workflow based on ontologies*, Technical report, Polelo Research Group, University of Pretoria, 2005.

[38] Wolfgang Emmerich, *Engineering Distributed Objects*, Wiley, August 2001.

[39] Wolfgang Prinz, Gloria Mark, Uta Pankoke-Babatz, *Designing Groupware for Congruency in Use*, Proceedings of the 1998 ACM conference on Computer supported cooperative work, 373-382,1998.

8.1. Web References

[40] *µCODE*, http://mucode.sourceforge.net, Last accessed May 2004

[41] *EtherYatri.NET*, http://www.geocities.com/siddharthuppal/EtherYatri.htm, Last accessed May 2004

[42] *Improving Gnutella Protocol - Protocol Analysis And Research Proposals*, http://www9.limewire.com/download/ivkovic_paper.pdf, Accessed 10 September 2005

[43] *JADE*, http://jade.tilab.com/, Last accessed May 2004

[44] *Microsoft SharePoint*, http://www.microsoft.com/sharepoint/overview.mspx, Last accessed 31 January 2006

[45] *MONADS*, http://www.cs.helsinki.fi/research/monads/, Last accessed May 2004

[46] *MSDN*, http://msdn.microsoft.com, Last accessed 22 Feb 2006

[47] MSDN, *What's New in the .NET Framework Version 2.0* , http://msdn2.microsoft.com/en-us/library/t357fb32.aspx, Accessed: 23 Feb 2006

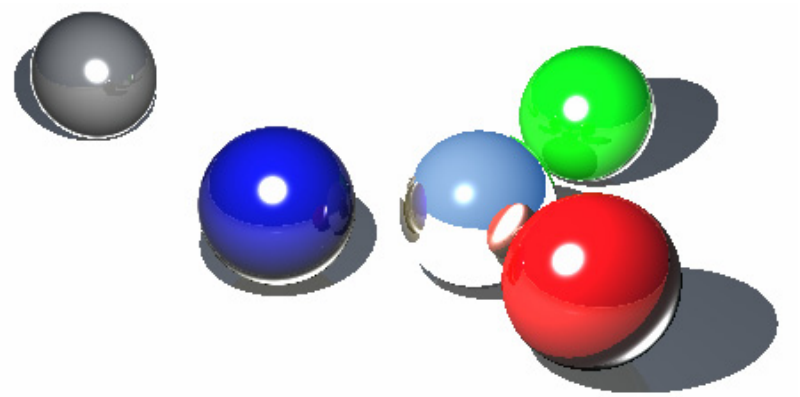
[48] *Novell iFolder*, http://www.novell.com/products/ifolder/ , Last accessed 31 January 2006

[49] *Sourceforge* , http://sourceforge.net/, Last accessed 31 January 2006

[50] *Subethaedit*, http://www.codingmonkeys.de/subethaedit/ , Last accessed 31 January 2006

[51] *The Gnutella Protocol Specification v0.4*, 1. Document Revision 1.2. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf , Accessed 10 September 2005

[52] *X-peers*, http://www.xpeers.net/ , Last accessed May 2004



The ray traced image that appears at start of each chapter was created by Jiten Rama. The colored spheres represent the states of a node in the simulator. These are red, green, grey and blue. The mirror sphere represents the reflective property of the propagation node to mirror requests of other nodes. Furthermore, the spheres represent the distributed characteristic of the users of the system.