AN ONTOLOGY-BASED P2P INFRASTRUCTURE TO SUPPORT CONTEXT DISCOVERY IN PERVASIVE COMPUTING

CHIN CHUNG YAU
(B.Eng.(Hons.), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2005

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisors, Dr. Zhang Daqing and Dr. Mohan Gurusamy, for their advice and encouragement throughout the research and the thesis writing process. I want to thank the Department of Electrical and Computer Engineering for offering me this possibility to pursue a Master degree in the Engineering field at the National University of Singapore. I also appreciate Institute for Infocomm Research for this opportunity to do research in the area of Pervasive Computing.

I would also like to thank my colleagues in I^2R including Dr. Jit Biswas, Wang Xiaohang, Yu Zhiwen, Aming, Thang, Sanka, Zheng Song, Ni Xiao, Thng Haw, Kailash, Dzung, Shen Tat, Chun Yong and Bryan. Not only have they provided me with useful feedback and suggestions on my work, they have also helped me to enjoy myself doing research in the institute, and made it a very fruitful experience for me.

Last but not least, I dedicate this work to my parents and siblings, as well as my girl friend Lay Keat, who have stood by me during these years, whose love and support have seen me through ups and downs in life. To all of you I want to say, I love you.

TABLE OF CONTENTS

SUMMARY

The advancement in today's computer hardware and software technologies have moved us one step closer to materialize the pervasive computing vision, the vision that computer systems, from embedded devices to large scale infrastructure, exist anywhere at anytime. Context-awareness is perhaps the most salient feature to turn such pervasive computing environment into smart space, where computer systems are able to exploit context of users, devices, and environment to offer value-added services that personalize application behaviors. In a smart space, embedded sensors and information sources form the pool of context information providers that offer plenty of context information. Through a process called context discovery, context-aware services and applications are able to find the suitable context information providers that can give the necessary context information to them. The existing context discovery schemes, however, are limited to functioning within a single smart space. This has greatly prohibited the proliferation of inter-space context-awareness in pervasive computing.

In this dissertation, we address the issue of context discovery in context-aware computing beyond single smart space. We propose a hybrid decentralized-centralized context discovery model, which leads to the design of a context discovery platform called *Orion*. In this model, all computing entities in a smart space are peers to one another, playing the role of both context provider and context requester simultaneously. A *Discovery Gateway* (DG) serves as the super-peer in a smart space, which is responsible to match a context provider to a context requester in the context discovery process. The DGs in different smart spaces form a peer-to-peer (P2P) ad-hoc message routing overlay network, known as the *Orion Network* (ONet). As a result, a lookup

query searching for context providers located in other smart spaces can be appropriately forwarded across the ONet to reach the relevant DG. To reduce the amount of duplicate messages as a result of the flooding-based message forwarding in the ONet, the DGs that share common interest in the context information they are registered with are clustered into a *Semantic Community* (SeCOM). As such, queries are only forwarded to DGs within a SeCOM that is able to resolve them. Simulation results reveal a significant reduction of duplicate messages in Orion compared to an overlay network that uses pure flooding search mechanism. On top of that, to promote interoperability between heterogeneous devices, we introduce a semantic matching technique in the provider-requester matchmaking procedure. This technique makes use of the class equivalence semantics inherited from the ontological description of the information.

This dissertation identifies the issue of inter-space context discovery, and presents Orion as the solution to the issue. The platform enables discovery and retrieval of context information from distant smart spaces, thereby allowing more flexible design of context-aware applications and more dynamic use of a wide range of context information from multiple sources. We believe that the achievement in inter-space context lookup and retrieval can overcome the single-space limitation of context usage in current literature, as well as foster new research initiatives that deal with wide-area context.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Overwhelmed with seamlessly integrated and interoperable embedded devices and services, pervasive computing applications need to be context-aware. This chapter introduces background on context-aware pervasive computing, followed by discussion of the motivation, goal and contribution of this research – a scalable context discovery platform for the context-aware computing systems.

1.1 Research Background

1.1.1 Pervasive Computing

Weiser unveiled the vision of *ubiquitous computing* (later also known as *pervasive computing*) more than a decade ago as the emerging model for the computing world in the 21st century [1]. In pervasive computing environment, massive amount of embedded computing devices and autonomic services gracefully integrate with human users, performing any task in an unobtrusive manner, such that their existence is taken for granted in everyday life. Using wearable mobile devices to control electronic appliances at home remotely, reading email from large display monitor mounted on the wall, issuing commands to machine with only hand gestures, monitoring home security alarm system from the office, and managing personal medical profile over the Internet, are merely a few of the exemplary scenarios that paint the picture of a pervasive computing environment. Compared to the current computing paradigm, pervasive computing sees the migration of computing from general purpose computers (e.g. desktop, workstation, mainframe) to customized mobile terminals (e.g. notebook, personal digital assistants, mobile phone, etc). It also exhibits the trend towards the

pro-active interaction among the computing devices and the surrounding system infrastructure, often without explicit control.

As a result, our living environment is transforming into a *smart space*. A space can be an enclosed area such as house, vehicle and office room, or it can be a well-defined open area such as campus, sports stadium and outdoor parking lots. Smart space brings together two disjoint worlds – computing infrastructure and physical infrastructure, and enables sensing and control of one world by another. The smart home environment, for example, is a smart space where all in-home appliances are connected, either through wired or wireless medium, and the functions of which can be automatically customized to an occupant's needs.

Pervasive computing smart space is a vision too far ahead of itself in the early 90's, and it is not until now in the 21st century that we are in a better position to pursue it. As wireless communication technologies, personal communication devices, feature-rich mobile terminals, and easily accessible network infrastructures develop rapidly, we now have the necessary technological platform to materialize the vision. Many projects were started since the late 90's. Some well known projects in the industry include, to name a few, the *DigitalHome*[1] at Intel, the *CoolTown*[2] at HP, the *Easy Living* at Microsoft [2] and the *Digital World*[3] at SAMSUNG. In the academic arena, we have the *Project Aura*[4] at Carnegie Mellon University, the *Oxygen*[5] at MIT, the Project GAIA [3] at University of Illinois Urbana-Champagne, the *AwareHome*[6] at

[1] The DigitalHome – Intel Corporation, http://www.intel.com/technology/digitalhome
[2] CoolTown – HP, http://www.cooltown.com/cooltown
[3] The DigitalWorld – SAMSUNG, http://www.samsung.com/HomeNetwork
[4] http://www.cs.cmu.edu/~aura
[5] http://oxygen.lcs.mit.edu
[6] http://www.cc.gatech.edu/fce/ahri

Georgia Institute of Technology, the *Portalano*[7] at the University of Washington, and many more.

1.1.2 Context and Context-Awareness

A minimally intrusive pervasive computing smart space has to be context-aware [4]. But what really constitutes a "context"? Oxford Dictionary defines "context" as "circumstances in which an event occurs". While this is a general definition, the term has been interpreted differently in computer science and engineering principle.

1.1.2.1 What is Context?

Everything in the world happens in certain context, and such context can be exploited in the computing world as implicit input to the computing systems [5]. It can greatly enhance the functionality of the computing systems in terms of decision making and output adaptation, shaping the smart space to become intelligent in reacting naturally and unobtrusively to human needs. Schmidt *et al*. define context as the knowledge about user's and IT device's state, which includes the state of the surroundings, situation, and location [6]. To be more general, Dey defines context as any information that can be used to characterize the situation of the inhabited entities (including person, computational object and environment) and the circumstances under which interactions between these entities take place [7]. The interpretation of context throughout this thesis is mainly based on the widely accepted Dey's definition of context.

Different category of contexts has been identified in the literatures. Schilit *et al*., in the notable work PARCTAB, divide the types of context into three categories, namely the

[7] http://portolano.cs.washington.edu

location of user, the identity of user, and the state of computing resources [8]. This, however, does not cover extensively all context types in a smart space. On the contrary, Dey classifies the context in a smart space to be the location (e.g. place, room number, post code, etc), the identity (e.g. user ID, preferences, personal information, etc), the activity (e.g. meeting, sleep, lunch, watching TV, etc) and the time (e.g. date, +GMT, time span period, etc) [7]. On the other hand, we may view a pervasive computing smart space as a contextual environment scattered with contextual object - user object, location object, computing entity object, and activity object. Each and every instance of these objects is associated with its very own context category [9]. For instance, given a person (i.e. user object), he may provide context such as personal profile, medical record, to-do activities, etc. Given a meeting situation (i.e. activity object), the meeting duration, number of participants, meeting venue, agenda, etc, are considered as its associated context.

1.1.2.2 What is Context-Aware Applications?

Since the notion of context-aware computing was introduced by Schilit *et al*. in 1994 [8], context-awareness has gradually become an essential element in ubiquitous computing [4]. It denotes the situation where an entity is cognizant of the context of itself, of its surrounding environment, and of the entities it is interacting with. Therefore, a context-aware system is able to interpret and adapt to the input context, and provides any relevant information or adaptive services to the user in response to the changing context [7].

We modified Lieberman and Selker's diagram in [5] that represents context to formulate the schematic view of a general context-aware application in Figure 1. Any

computing application, including the context-aware application, can be abstracted as a black box that generates various kinds of outputs depending on the input to the system.

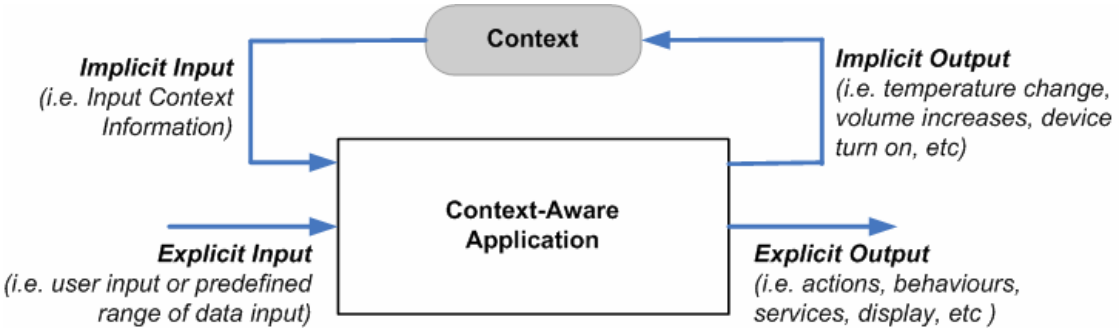


Figure 1. Context-Aware System Model

A traditional computing application would only accept *explicit input* that is presented by the user (e.g. keyboard typing, mouse clicking, gesture, etc), or by a pre-defined set of input data (e.g. spreadsheet, files, functional parameters, etc). After processing, *explicit output* is generated, that includes displaying information, performing actions, and providing services. The application model is expanded in the context-aware computing, where context information contributes as the *implicit input* to the computing black box and becomes part of the processing parameters. That is, the application now can decide what to do based on the explicitly presented input and the context. As a result, not only the explicit output is well adapted to the context, but the output may also iteratively alter the state of context in the form of *implicit output*.

The context-aware application model has offered a wide range of context-aware applications and features. [8] describes 4 classes of context-aware applications, namely:

- Proximate selection of nearby object with user-interface techniques

- Automatic contextual reconfiguration of object components via adding, removing and altering actions
- Contextual information displays and commands issuing according to the context in which they are issued
- Context-triggered actions based on IF-THEN rules to specify the adaptation behavior

Opposed to the above class category, Pascoe proposes taxonomy of context-aware features, including *contextual sensing*, *contextual adaptation*, *contextual resource discovery* and *contextual augmentation* [10]. Dey combines these ideas and lists three general categories of context-aware features that a context-aware application may support: presenting information and services to a user, automatic execution of a service, and tagging context to data for later retrieval [7]. The first category, *Context Presentation*, denotes the application that displays context information to the user. The second category, *Context Execution*, indicates the ability to execute an action or modify a behavior based on the changing context. The third category, *Context Tagging*, associates data with related context so that the data can be viewed when the user is in that context.

A few examples of context-aware applications are listed below. Each application is classified according to Dey's 3-category classification of context-aware features:

- Changing cell phone functional behavior automatically based on combination of sensed context [11] – (category 2)
- Presenting localized exhibition information to visitors based on visitors' location and preference [12] – (category 2 and 3)

- Selecting appropriate network channel for establishing communication based on service availability and bandwidth requirement [13] – (category 1 and 3)
- Routing an incoming phone call to a fixed-line phone that is nearest to the call recipient's current location [14] – (category 2)
- Guiding office visitors with directional map instructions and meeting schedule [15] – (category 1 and 2)

1.2 Motivation

Context-aware smart spaces are rich in context information, ranging from low-level basic context such as temperature, noise level, device status, weight, and location coordinates, to high-level complex context such as activity schedule, medical profile, relations between people, user preference and road traffic condition. In terms of context information processing, we broadly classify the entities participating in a context-aware smart space into two categories: the *context provider* and the *context requester*.

A *context provider* is any entity that supplies context information. Environment sensors, information sources, monitoring software and context knowledge base, for example, are categorized as the context provider. A *context requester* is any entity that consumes context information for its context-aware processing. Examples of context requester include context-aware applications and services, context-sensitive agents and context processing operators. A single computing entity can take up dual roles as a provider or a requester at different time, for different tasks. For example, a mobile phone may, at one hand, act as a context requester who modifies its profile settings

automatically based on different input situational context; while on the other hand, be a context provider revealing the user's current location.

The existence of both providers and requesters can be in one of the two forms: co-existing in a single device, or existing independently from one another [16]. The first form of existence results in the sensors (i.e. context provider) being embedded onto the same device the context-aware application (i.e. context requester) is residing on. For example, handheld devices are often integrated with motion sensors to capture gestures and device orientation information for graphical user interface adaptation (see [16], [17] and [18]). The second form of existence includes context-aware applications that can acquire context from external sources, either from independent sensors (e.g. temperature sensor, location beacon, application peers, etc) embedded in the smart spaces, or from the context infrastructure (e.g. Gaia [3], Context Toolkit [19], Context Fabric [20], Solar [21], CoBra [22], Semantic Space [9], etc) that handles the acquisition, interpretation, storage, and dissemination of context information. Figure 2 outlines a scenario of the second form of existence, where context information is constantly flowing from *m* context providers to *n* context requesters, whose existence is independent from one another.

Due to the drawbacks in the first form of existence (e.g. hardware constraint, limitation on sensor type, battery level, accuracy, etc) and the flourishing of embedded sensors in the pervasive computing smart spaces, the second form emerges as the preferred channel for context-aware applications to acquire context information. This ensures greater flexibility in system design, and more variety of context information can be manipulated at the same time. Consequently, context-aware applications can be rapidly developed, while context sources can be easily deployed.

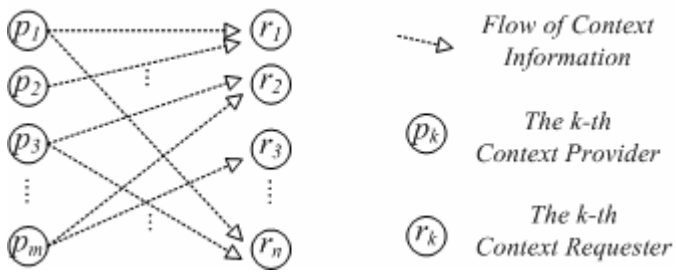


Figure 2. Context requesters acquire context information from different context providers that exist independently from one another

However, smart spaces are overwhelmed with heterogeneous and volatile context resources (i.e. both context provider and requester). It is not feasible and not scalable for an individual application to maintain connections to the sensors and information sources statically or via pre-defined setting. Such static connectivity approach is especially undesirable for resource-constrained devices with low memory capacity, low processing power, and low communication capability.

To ensure dynamic connectivity and flexible use of context information from multiple sources, the context requesters need to automatically locate the appropriate set of context providers which can produce the desired and necessary context information [4]. Such discovery process is known as *context discovery*. "Discovery" is recognized as a fundamental operation for determining the global state of a distributed system with minimal user intervention in the process [23]. Similarly, context discovery allows appropriate context information to be located and retrieved from a set of independent context providers scattered in the pervasive computing smart spaces. Therefore, context discovery enables a context-aware application to gain access to and to adapt to the broad spectrum of dynamic context information without prior knowledge about the respective context providers.

The current work in context discovery (e.g. [19], [21], [24], [25]) has been focusing in the discovery of context resources within a single smart space. However, the need to scale context discovery across different smart spaces remains relatively unexplored. The need for inter-space context discovery is supported with the following 3 observations:

- *Observation 1*: We observe that, types of context in different category of smart spaces can be very diverse. In home smart space, for example, context information is related to family activities, relationship of family members, placement of devices, and state of electronic appliances. On the other hand, context generated in vehicle smart space includes driver status, location within city, relevant distance to approximating objects and conditions of various elements in the vehicle. Therefore, the type of context information a provider produces to a large extend depends on the smart space it is residing in or associated with. For instance, it is unlikely that John's working schedule can be found in his car's engine monitoring system; similarly, it is inappropriate to find road traffic condition from any of the sensors within a house smart space.

- *Observation 2*: As a context-aware application moves from one space to another (e.g. from building level 1 to level 2, from house to office, etc), it can be cognizant of contexts in both the "been-to" spaces, as well as the "going-to" spaces. For example, an individual's health status measured by the various heterogeneous ubiquitous sensors in the smart spaces he/she has been to is an essential input for a context-aware healthcare advisor system in generating relevant healthcare advices from time to time. On the other hand, the current status of the printing service and the network access service in the spaces a

person is heading to, for instance, is required for his/her laptop to decide on where and how to print a document upon arrival.

- *Observation 3*: Context provider of specific context information of interest can be ubiquitously available in different smart spaces. For instance, a medical officer, upon an emergency medical treatment, needs to acquire the patient's medical profile that is stored in his home gateway, and to retrieve his hospitalization records possibly maintained by different hospital web databases.

These observations bring forward the need for *inter-space context utilization*, i.e. deriving and retrieving context of different smart spaces, possibly provided by context providers residing in other spaces. Figure 3 provides a schematic overview representing the utilization of context information via inter-space context retrieval.

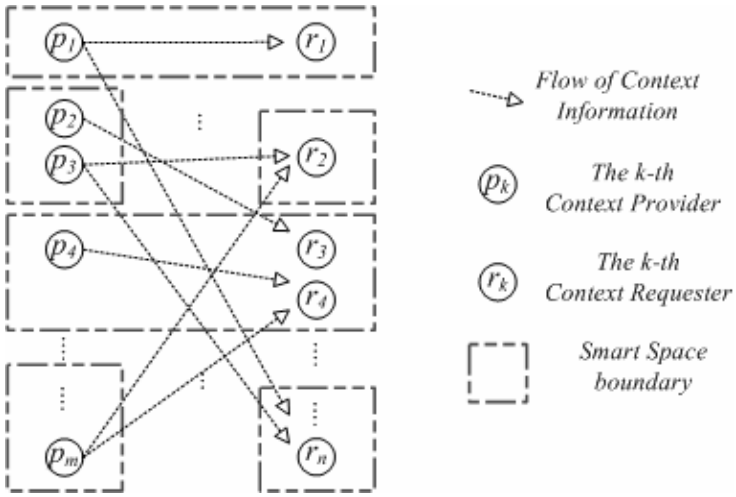


Figure 3. Inter-space context utilization

The observations mentioned above outline a few of the scenarios for context requesters to locate different context information from different smart spaces. As we will be explaining in Section 2.3.6, the existing context discovery schemes can hardly

perform well when dealing with inter-space context discovery, due to the limitation in their architecture design meant for single space functionalities. Consequently, context discovery across various smart spaces needs to be addressed as well. Therefore, we anticipate a context discovery platform that can enable the lookup of context beyond local smart space boundary.

1.3 Objectives

In this thesis, we focus on the issue of inter-space context discovery. After analyzing related work, we realize that current approaches and protocols do not scale well to handle context discovery across many smart spaces. As a result, we propose a Context Discovery Platform, called *Orion*, to fulfill this purpose. Orion is a set of context discovery protocols operating on a peer-to-peer infrastructure, which is capable of mediating context requester with the relevant context providers regardless of their localities in space. Orion allows context publishing and context lookup to take place, thereby facilitating the discovery of context information. Context providers, such as sensors and information sources, can advertise about their existence in Orion; while context requesters, such as context-aware applications, can easily locate the necessary and appropriate set of context providers by querying Orion.

1.4 Research Challenges

The *scalability* of inter-space context discovery platform needs to be ensured. Discovery across many smart spaces implies that the platform needs to accommodate large number of sensors, devices, applications and users. The nature of pervasive computing dictates that these entities can join and leave the spaces, and traverse both geographical as well as network boundaries, at anytime, anywhere. On top of that, it is

essential to have performance scalability, so that query processing and resource utilization remains efficient as the system size increases. Besides that, it also needs to handle huge information processing load as and when it is necessary.

Device and service *interoperability* must be addressed as well. Different versions, vendors, specifications, and standardizations may cause serious interoperability issue when these devices and services are to interact with one another. There are two key elements to successful interoperation. First, a common representation model needs to be established to represent the context information, so that any two autonomous computing entities can communicate with one another. Various context modeling techniques have been established, for example [22] and [9] use ontology modeling and reasoning over context information, [26] proposes a context modeling language similar to entity-relations UML modeling adopted in the object-oriented computing, Gaia uses prolog-based context predicates [27], and Solar adopts key-value attribute pairs [21].

After ensuring the devices and services share a common vocabulary in publishing the context information, they then need to understand the semantics of the vocabulary. For example, context descriptions `<location = washroom>` and `<location = toilet>` share the common semantics, although they are different in their syntactic labeling. The devices and services need to be equipped with semantics reasoning techniques in order to achieve interoperability at the semantics level. This become the second key element to interoperability.

1.5 Contributions

The areas of research that are being identified and addressed in this thesis include architectural support for inter-space context discovery, peer-to-peer infrastructure for

query distribution, and context modeling for the resource matchmaking. The contributions of this dissertation are summarized below:

- A generic architecture for context publishing and lookup that is scalable across different smart spaces
- A query forwarding mechanism for efficient context lookup using P2P-based semantic overlay network techniques
- An ontology-based context modeling for meta-context representation and resource matchmaking using Semantic Web ontology modeling and reasoning technologies.
- A development framework that gives leverage to context-aware application developers.

1.6 Outline

The thesis is structured in the following way. Chapter 2 provides introductory overview about the Peer-to-Peer computing system and the Semantic Web, the two technologies that Orion is based on. Then, the various related work in context discovery is reviewed, and their ability to support inter-space context discovery is highlighted.

Chapter 3 reveals the insights into Orion context discovery platform. First, the different context discovery models are introduced. The hybrid centralized-decentralized model presents the model that Orion is based on. Following that, the architectural overview of Orion is presented. The key elements in Orion, namely the Discovery Gateway, the P2P message forwarding overlay network and the ontology-

based matchmaking procedure are put together to support the context discovery operations that made up of context publishing and context lookup.

In Chapter 4, the details of the P2P network infrastructure in Orion are covered. The concepts of Orion Network (ONet) and Semantic Community (SeCOM) are established, and a set of algorithms is derived to maintain and to support the various network operations, especially the search mechanism in Orion. The P2P network infrastructure is evaluated via simulation. The results are analyzed at the end of this chapter.

Chapter 5 looks into the matchmaking procedure in Orion. The ontology-based advertisement template, as well as the corresponding query language, is presented in details. Based on the advertisement and the lookup query specification, the semantic matching technique is derived and introduced.

The prototype architecture of the Discovery Gateway is presented in Chapter 6. This chapter also reports the results of query response time analysis based on the overlay network constructed on the public TCP/IP network infrastructure using the Discovery Gateway prototype.

The conclusion in Chapter 7 summarizes the contributions made in the thesis. Future research directions are listed as well.

CHAPTER 2 BACKGROUND AND RELATED WORK

In this chapter, we look at some of the technical ground that Orion is based upon, namely the Peer-to-Peer Network, and the Semantic Web ontology modeling and reasoning techniques. We also examine the various related work on context discovery.

2.1 Peer-to-Peer Network

Peer-to-peer (P2P) network has become one of the fastest growing and most popular Internet applications over the past few years. In this section, we provide a brief overview of P2P network systems, and look into the decentralized search mechanisms in the unstructured-based P2P network.

2.1.1 P2P Overview

A peer-to-peer (P2P) network does not have the notion of clients and servers. Each peer node in the network simultaneously functions as both client and server to the other peer nodes. Comparing to the traditional client-server model, such as FTP file sharing and webpage servers, P2P computing model decentralizes the traditional centralized model to the distributed service-to-service model.

As described by Roussopoulos *et al*., P2P network exhibits three characteristics: *self-organization*, *symmetric communication* and *distributed control* [28]. P2P network is *self-organized*, because there is no global directory that dictates the connection between any two peers. The network is formed in an ad hoc manner through the peer discovery process. Overlay communication channel is laid between two peer nodes, and the channel is *symmetrical*. Information can flow in two directions, depending on

whether the peer node acts as the content provider or requester. Finally, the course of action and behavior of each peer node is *independently controlled* without any central controller.

P2P research can be divided into 4 groups – search, storage, security and applications [29]. Among them, the search capability of a P2P system is leveraged in Orion. Search methods in P2P network can be either centralized or decentralized. The centralized approach requires the use of a centralized directory service. In decentralized approach, P2P network is broadly classified into unstructured-based P2P and structured-based P2P, based on the P2P overlay topology setting and the placement of the resources.

In the coming sections, the various search mechanisms devoted for each of the P2P network type are examined and compared. The term "resource" is used in this section to commonly denote the items (e.g. files, contents, services, etc) being provided and requested by the peers.

2.1.2 Centralized Search in P2P Network

In this search approach, a centralized search facility is established to keep track of the index to the resources available in the peers. Although queries to search for relevant resources are resolved by the central server, communication between peers during the resource retrieval is performed in a P2P manner. The first widely successful P2P file sharing system that employed the centralized lookup approach is Napster[8]. Skype[9], a voice-over-IP Internet telephony system, also adopts such centralized P2P communication model.

[8] Napster, http://www.napster.com
[9] Skype, http://www.skype.com

The centralized search architecture offers powerful and responsive query processing, allows easy management (e.g. user login, billing, resource monitoring, etc) and inherits the scalability and flexibility properties of the P2P network. However, the central needs to handle high query load, and remains as a single point of failure. From a commercial standpoint, centralized approach requires a sizable capital investment in the infrastructure as well. Consequently, most recent P2P search methods have adopted the decentralized search architectures.

2.1.3 Decentralized Search in Unstructured-Based P2P Network

In unstructured-based P2P network, the overlay connections between the peer nodes are random, i.e. no fixed topology or node placement policies are applied in establishing the communication links. Each node discovers its own sets of neighbouring nodes, and forms the one-hop neighbourhood. While each node holds its own limited set of resources, query for locally unavailable resources can be searched among the neighbours. The queries are relayed from one node to another, until the resource is found, or until the forwarding TTL (time to live) expires.

In Gnutella[10], the resources are only indexed by the peer that caches them, and query for the resource can be resolved by probing at the proper peer. The peers are probed using pure flooding mechanism, i.e. query is forwarded to all neighbouring peers if it cannot be resolved locally. Gnutella marks the birth of flooding-based query distribution in unstructured P2P network, no doubt offering many rooms for improvement for its heavy network traffic, high message redundancy and inefficient probing mechanisms.

[10] Gnutella, http://www.gnutella.com

As a result, various heuristics in the forwarding strategies are proposed. One way is to minimize the number of hosts that has to be probed whenever an unresolvable query needs to be forwarded (i.e. heuristic in forwarding strategy). Freenet[11] uses *random walk* technique, whereby a query is only sent to one randomly selected neighbour. Lv *et al*. extends the technique to *k-walker random walk*, which means at one time *k* random neighbours are selected instead [30]. Furthermore, to increase the likelihood of response from a random neighbour, [31] and [32] used *biased random walk*, where their selected neighbours are those with higher flow capacity and higher outgoing degree respectively. Other heuristics include Directed Breadth First Search (Directed BFS) technique, where each node maintains simple statistic on its neighbours, and queries are only forwarded to neighbours that have produced many quality results in the past (e.g. returning the most results, processing query with shortest message queue, etc) [33]. Rather than "*who to send*", *expanding ring* decides on "*how far to send*" by successively broadcasting queries to neighbours with an increasing TTL in each successive iteration [30]. Such method is also known as *iterative deepening search* [33].

To improve heuristic in routing decision, Crespo and Garcia-Molina introduces *Routing Indices* (RI) that provides "hint" as to which "direction" can better lead to the destination node [34]. Given a query, RI returns a list of neighbours ranked according to their *goodness* for the query, as measured by the number of documents found in a path. Similar to RI, Yang and Garcia-Molina propose to use *Local Indices* for indexing over data of all nodes within *r* hops [33]. Thus, a node can process the query on behalf of every node within *r* hops. Instead of indexing the actual data, Rhea and Kubiatowicz present a *probabilistic location algorithm* that associates a probability of

[11] Freenet, http://freenet.souceforge.net

finding a document in each neighbour with the use of the *attenuated Bloom filters* [35]. Probabilistic information about the location of content can also be specified by *Exponentially Decaying Bloom Filter*, which encodes the content hosted by all neighbours for each forwarding direction [36].

Some researchers propose heuristic in the peer neighbourhood formation. Semantic Overlay Network (SON) clusters peer nodes that share semantically related resources into a sub-overlay network [37]. Queries are only broadcasted within SON that is able to answer them. *Acquaintances* [38] applies similar approach, but semantic relations are discovered spontaneously at runtime, without having to explicitly classify the resources compared to SON. DiCAS [39] labels each cluster from number 1 to M, and all peers in the same cluster cache response to query where the equation - `cluster ID = hash (query) Mod M` is satisfied. Subsequently, queries are only forwarded within cluster of which the group ID matches the hash value of the query. To organize the peers in the semantic cluster, RATTAN adopts tree-like logical structure [40]. Query destined to a specific cluster is always issued to the root of the associated tree overlay network, and then transmitted down the tree towards the leaves. FloodNet, on the contrary, proposed to organize unstructured P2P network into multiple tree-like low-diameter clusters, and forward the messages using the *LightFlood* technique [41]. Instead of clustering, Sripanidkulchai *et al*. explore *interest-based locality* (i.e. if a peer has a piece of information that another peer is interested in, it is also likely to have other information that is of interest), and establish *interest-based shortcut* between the peer nodes that share similar interest locality. [42].

Unstructured P2P network also faces the issue of topology mismatching [43]. Two neighbouring peers may actually be placed far away in the low level physical network.

To overcome the problem, the unstructured P2P network topology has to be adaptive to the underlying physical network. Landmarking technique is introduced [44] where all nodes at bootstrap locate the *landmark* node of a *bin*, and measure distance (i.e. round trip time (RTT)) to *landmark*. Peer subsequently decides to join the *bin* where all nodes in the same bin are physically close to one another. mOverlay [45] proposes to use dynamic landmark instead, where the group ID of each peer group is the landmark itself. Peer groups are formed by peers that are physically close to one another. A joining node will locate a dynamic landmark that is the closest to itself and join the group where the landmark belongs to. Instead of relying on landmark, Liu *et al*. introduce *Location-aware Topology Matching* (LTM) [46]. Each node actively probes its one-hop and two-hop neighbour for the latest communication RTT (i.e. TTL2 probing), and chooses to disconnect peer with poor RTT response during runtime. Iteratively, this ensures all paths are within the shortest distance (in terms of latency delay).

While different kinds of heuristics are proposed, another form of unstructured P2P network has emerged - the super-peer P2P Network. A super-peer is a peer node that acts as a centralized server to a subset of client peers [47]. These client peers submit queries to and receive results from the super-peer. Super-peers are connected to one another in a P2P manner, forming the P2P message routing overlay network. They are responsible to route messages over the overlay network and answering queries on behalf of the clients. The super-peer network model is adopted in the Gnutella2[12] network.

[12] http://www.gnutella2.com

2.1.4 Decentralized Search in Structured-Based P2P Network

In structure-based P2P network, the P2P overlay topology is tightly controlled and the placement of contents/files is not random but is determined at specific locations. This tightly controlled overlay topology structure enables the P2P systems to resolve query very efficiently by limiting the searching hop within a bounded number of hops.

Structured-based P2P network typically support distributed hash table (DHT) functionality in mapping key to node, i.e. the lookup operation returns the identity of the node storing the resource associated with the key. The notable structured-based P2P networks include Chord [48], Content Addressable Network (CAN) [49] and Pastry [50]. In these systems, each node is responsible for storing a range of keys and the corresponding resources. The nodes are connected into an overlay network with each node knowing several other nodes as neighbours. Chord organizes the nodes into a ring network topology, while nodes in CAN are arranged as a virtual d-dimensional Cartesian coordinate space on a d-torus. When a lookup request is issued from one node, the message is routed through the overlay network to the node responsible for the key. As for Pastry, replication of published resources is placed on nodes which the ID of nodes is the closest in the ID namespace of the resource, and prefix addressing routing is used. As a result, Chord, CAN and Pastry guarantee lookup to be accomplished within $O(\log N)$, $O(N^{1/d})$ and $O(\log_{2^b} N)$ hop counts respectively (N is the total number of nodes, d is the dimension value and b is the configuration parameter).

While DHT-based P2P systems show efficient lookup and failure resilience, they exhibit certain drawbacks. Only single-key based lookup is supported in DHT, and multi-attribute key and range queries are not allowed. This affects the flexibility in

formulating expressive query, especially when generating a precise query. Furthermore, excessive overhead is needed to maintain the overlay network when dealing with transient peers. Different degrees of topology restructuring and resource redistribution are required whenever any peer joins and leaves the system.

2.2 Semantic Web Ontology Modeling and Reasoning

To date, information on the World Wide Web is designed merely for human reading, but not for computer programmes to manipulate meaningfully, i.e. computers have no way to process the semantics of the web contents. The Semantic Web turns the table by bringing meaningful structure to the content of the Web pages.

Semantic Web is defined as "the conceptual structuring of the Web in an explicit machine-readable way" [51]. Semantic Web aims at enabling computer machines with the capabilities to "understand" the semantics of web content, and therefore allowing machine to process them automatically in cooperation with other machines and users. Marshall and Shipman summarize the three visions of the Semantic Web [52]:

1. Semantic Web organizes the loosely connected networks of digital documents that make up the Web.

2. Semantic Web creates a networked knowledge ontology that allows knowledge to be acquired, represented and utilized.

3. Semantic Web offers an infrastructure for sharing of data and knowledge developed and distributed by different domain-oriented applications.

To realize Semantic Web, computer machine first needs to represent web content as knowledge, and subsequently needs to interpret its semantics. W3C has initiated a set

of knowledge representation standards. Figure 4 outlines the layer model of knowledge representation language in the Semantic Web.

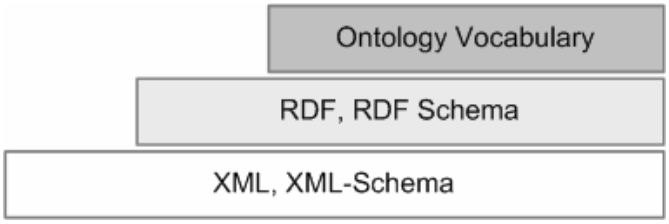


Figure 4. The Semantic Web layer language model, where each layer is building on the layer below

The foundation of knowledge representation is the *eXtensible Markup Language* (XML). XML has been widely adopted in today's Web as flexible information markup language, in which the grammars are described in the XML-Schema. However, XML and XML-Schema only allow specification of syntactic conventions, but do not impose semantic constraints on the meaning of a document.

Based on XML syntax, the *Resource Description Framework* (RDF) defines a data model to represent data's machine-processable semantics, making interoperable exchange of semantic information possible between the machines [53]. RDF is expressed in a (*subject*, *predicate*, *object*) triple, where each triple outlines the relation property (i.e. *predicate*) of a resource (i.e. *subject*) to an *object*, which can be either another resource or certain value. RDF Scheme [54] lets developers to define particular vocabulary for RDF data and specify relationships between properties and resources.

Semantic Web uses ontology to present heterogeneous semantic information. Ontology is an explicit, machine readable specification of a shared conceptualization in terms of entities, relations, instances, functions and axioms [55]. Ontology

vocabulary requires an expressive language, such as the Web Ontology Language (OWL) [56] (a W3C's recommendation for ontology language). Based on the RDF and RDFS framework, OWL is a knowledge representation language for defining, instantiating, interpreting, and reusing ontology knowledge. It adds formal vocabulary for describing concepts and their properties, such as equivalence, disjoint, transitive, symmetric, functional and inverse property to one another.

With the language model and the relevant knowledge reasoning tools, software agents are able to understand the semantics of the Web content and to interact intelligently to one another and to the users. Web resources can be defined and relations between resources, terms, and properties can be established. The ontology language can be further analyzed for consistency and inferences can be made. Consequently, inconsistent facts can be reconciled, while implicit facts can be discovered. The use of OWL-DL, for example, enables semantic reasoning of the concepts and relation properties to be performed via the Description Logic reasoning features.

Semantic Web technologies are not limited to the Web, and context-aware computing is one area where these technologies can be exploited. OWL is expressive enough to model the rich feature of context information and contextual entities in the smart spaces. It promotes knowledge sharing and reuse, and interoperates between the heterogeneous context resources at the semantic level. Ontology-defined context can also support expressive query and automated inference with its explicit semantic representations. Therefore, the use of Semantic Web tools (e.g. inferencing engine, Knowledge Base storage, etc) facilitates different management and processing tasks for the context-aware applications in acquisition, interpretation and dissemination of context information. A few example of context-aware systems that leveraged the

Semantic Web technologies include the SOUPA [57], Semantic Space middleware [9], Semantic e-Wallet [58], Task Computing Environment [59] and InforMa [60].

2.3 Related Work in Context Discovery

Context discovery is a key feature in many context-aware system infrastructures (i.e. known as "context infrastructure") that provides architectural supports for developing and deploying context-aware applications. We first present a brief overview of the various context infrastructures, highlighting the approaches taken for supporting context discovery. We then analyze these approaches, especially on their ability to scale context discovery across many smart spaces.

2.3.1 Context Toolkit

The Context Toolkit [19] developed at Georgia Institute of Technology is one of the pioneer context infrastructures that support systematic and rapid building of context-aware applications, by hiding away the complexity of the sensing and gathering of context information. It introduces four categories of components in a context-aware system: *Context Widget*, *Context Aggregator*, *Context Interpreter* and *Context Discoverer*. *Context Widget* enables applications to access to context data sensed by sensor, *Context Aggregator* merges different streams of related context data for representing context information related to specific entities (e.g. user, devices, environment, etc), and *Context Interpreter* interprets the raw context data into high-level context. For context-aware application to discover the different components, the *Context Discoverer* is deployed. *Context Discoverer* is a centralized directory system that registers the existence of the various components available for use by applications.

Applications can find a particular component with a specific name (i.e. white page lookup), or with a set of matching attributes (i.e. yellow page lookup).

2.3.2 Gaia Context Infrastructure

Gaia [3] is a middleware infrastructure for smart spaces, where physical spaces and the ubiquitous computing devices available in smart spaces are converted into a programmable computing system. The Gaia extension for context-awareness, i.e. *Gaia Context Infrastructure* [27], enables computer agents in smart spaces to easily acquire context information from the different distributed context providers. Context providers can advertise the set of context they provide to the *Context Provider Lookup Service*, so that they are discoverable by the agents. Context is represented as *context predicate*, specified using the DAML+OIL ontology language, such that the name of the predicate is the type of context being described. The advertisement is in the form of first order expression, and the matching between advertisement and the context predicates set is performed in the Lookup Service.

2.3.3 Solar

Solar [21] is a Context Fusion Network (CFN) infrastructure for context aggregation, composition and dissemination. Solar is formed by a distributed set of event operators that at one end connects to the data sources (i.e. sensors) while the other end to the data sinks (i.e. applications). Sensed context information is pushed into the Solar via one of the operators as an *event*. An event operator accepts one or more events, aggregates them based on predefined operator functions, and pushes the aggregated event (i.e. high level context) to the input of another event operator. Solar introduces *name advertisement* [61], a naming service for the data sources by using a set of descriptive attribute-value pair. The advertisements are stored in a directory service

based on Intentional Naming System (INS) [62], which composes of a distributed, self-configuring overlay network of name resolvers. It provides attribute-based registration and lookup interfaces. The data source for relevant context information is therefore discovered by name pattern matching in the resolver name space.

2.3.4 Strathclyde Context Infrastructure

The Strathclyde Context Infrastructure (SCI) [63] deploys *Context Server* in a Range (i.e. a similar notion for "smart space") to manage the distributed *Context Entities*, which are software components for representing entities (e.g. people, software, places, devices, etc) in a Range. Context information associated for each entity is represented as the entity's configuration, an event subscription graph between the entities. The Context Server also plays the role of *Context Trader* (similar to the concept of *Service Trader*) that can accept a request for context information and return a list of possible configuration based on behavioral specification matching techniques and automatic semantic reasoning about the configuration of each entity [25]. Such context discovery mechanism is performed based on the component trading approach.

2.3.5 Context-Aware Applications Platform

A Context-aware application platform is proposed by Efstratiou *et al*. [24] to support adaptive mobile applications to adapt to changes in the environment context. Mobile context-aware applications expose their adaptive mechanism to the platform with *adaptation policies* specified by the users. When context changes are detected and updated in the *Context Database*, the *Adaptation Control* coordinates the coexisting applications according to changes of the context. To locate the services that provide

the relevant contextual information, the platform relies on the UPnP architecture[13]. A service describes itself using an XML description template, outlining the service category, the access points for communications, and the information exchange format. Advertising of services is performed using broadcast announcement. The platform discovers the services, and receives notification events when the contexts of the services change.

2.3.6 Discussion

Context Toolkit, Gaia Context Infrastructure and SCI are using central repository for handling context discovery in a smart space. The centralized directory architecture is not scalable to handle large data volume and high query load, but unfortunately these are essential when we are dealing with wide-area context management. Although centralized server allows easy management and normally enjoys efficient query processing performance, it faces the risk of single point of failure. Consequently, centralized directory approach is not an ideal architecture for inter-space context discovery.

On the other hand, Solar adopts the decentralized approach by using distributed namespace resolver directory service based on the Intentional Naming System (INS). Architectural wise, a decentralized approach scales well to handle inter-space context discovery. However, each resolver in the INS needs to maintain an identical copy of the hierarchical representation of Solar's naming description, which results in constraining INS to support only limited range of service lookup.

Efstratiou *et al.*'s Context-aware application platform adopts the broadcast-based UPnP service discovery, which clearly lacks the scalability to make announcement

[13] http://www.upnp.org

beyond the local network boundaries. On top of that, when multiple context providers constantly broadcast about their existence, the network can be easily congested with broadcast messages. The frequency of broadcasting can also affect the lookup efficiency of a context requester. Clearly, broadcast-based approach is inappropriate to support inter-space context discovery.

In terms of representation model, all except Gaia adopts the keyword-based attribute-value context representation. Matching techniques are therefore constraint to string-based matching, and this could lead to semantic conflicts as identified in [64]. Resource interoperability among heterogeneous resources would need to be carefully dealt with by strict standardization on the names of the attributes and the range of the values for each attribute. Orion overcomes semantic conflicts by applying ontological description as semantic representation of the context resources, and by adopting semantic-based pattern matching for the matchmaking process.

2.4 Chapter Summary

In this chapter, background information about peer-to-peer (P2P) computing and Semantic Web, as well as related work in context discovery are presented. The readers are provided with a comprehensive survey about the variety of search mechanisms in P2P network, and the introductory overview about ontology modeling and reasoning techniques in the Semantic Web. The review of various related work outlines the different context discovery approaches in current context-aware computing research. The lack of inter-space context discovery support in the current approaches draws the needs for an inter-space context discovery platform, such as the Orion infrastructure. The Orion infrastructure is introduced, analyzed, and evaluated in the subsequent chapters.

CHAPTER 3 ORION: CONTEXT DISCOVERY PLATFORM

Orion is a context discovery platform dedicated for pervasive computing smart spaces. Context requester can rely on Orion to locate the necessary context provider, regardless of its locality. In this chapter, we first look at the context discovery model in general, and what constitute the requirements of a context discovery platform. Following that, the Orion architecture is presented.

3.1 Context Discovery

Context discovery is the process of automatic locating the whereabouts of the necessary context providers that are able to provide the desired context information [19]. It involves the interaction between three entities in the smart spaces, namely the context provider, context requester, and the context discovery platform. A general model that describes the interaction within and among these entities is introduced.

3.1.1 Context Discovery Model

Figure 5 outlines a general model that depicts the interactions between different entities in the context discovery process. These entities include the set of context provider $\boldsymbol{P}$, the set of context requester $\boldsymbol{R}$, and the context discovery platform $\boldsymbol{C}$.

A context provider $\boldsymbol{p_x}$, where $\boldsymbol{p_x} \in \boldsymbol{P}$ and $\boldsymbol{x}$ =1, 2, …, $\boldsymbol{m}$, is any entity in a smart space that supplies context information. Each $\boldsymbol{p_x}$ operates one or more *context generating function* $\boldsymbol{f_{p,x}}$: $\Delta \boldsymbol{c} \rightarrow \boldsymbol{I_x}$, which denotes that the generated set of context information $\boldsymbol{I_x}$ is an abstraction of the context of happening $\Delta \boldsymbol{c}$. Environment sensors, information sources, monitoring software and context knowledge base are examples of $\boldsymbol{p_x}$.

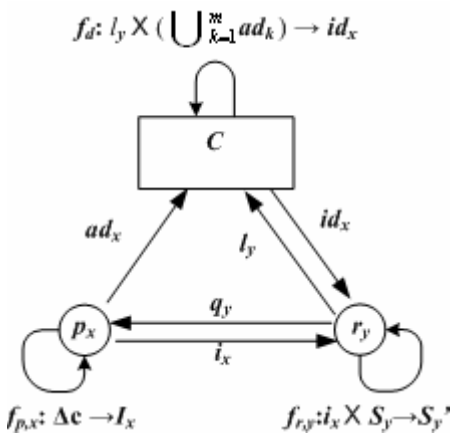


Figure 5. Context discovery model involving the context provider, context requester and context discovery platform.

A context requester r_y, where $r_y \in R$ and $y = 1, 2, ..., n$, is any entity that consumes context information for its context-aware processing. The consumed context information I_y' may be provided by one or more context providers, such that $I_y' \subseteq \bigcup_{x=1}^{m} i_x$, where $i_x \in I_x$. i_x is the context information provided by the context provider p_x. For every input i_x from provider p_x, the context-aware process function $f_{r,y}$: $i_x \times S \rightarrow S'$ carries out the context-aware processing that changes the requester's state from S to S'. A change of system state can be the adaptation of the system behavior or outputting of relevant context sensitive information [7], as well as waiting of another set of input context information. Examples of r_y may include context-aware applications, context-sensitive agents and context processing operators.

In ubiquitous computing, the number of context providers and requesters can be finitely large. To prevent static configuration of connectivity between p_x and r_y, the context discovery platform, C, can play the role of mediator between them. C is a set of protocols and necessary infrastructure to handle the discovery of context information. It allows r_y to locate the necessary and appropriate set of resource providers P', where $P' \subseteq P$, with minimal user intervention in the process. A p_x can

publish its metadata-based *context advertisement* ad_x to C for advertising the availability of its I_x. On the other end, r_y can look up the relevant P' from C by submitting a *context lookup query* l_y delineating the provider discovery requirements. The executing ground of C is the *context discovery function* $f_d : l_y \times AD \rightarrow id_x$. f_d matches the submitted l_y against the set of published advertisement AD, where $AD \subseteq \bigcup_{x=1}^{m} ad_x$, and identifies the registered context provider that can satisfy the needs of the requesting context requester, through the process known as *matchmaking* [65], [66]. Upon successful matchmaking, the identity id_x of the matched p_x is returned to r_y, carrying the access protocol information such as provider's IP address and port number. Upon a successful discovery, r_y can establish communication channel with the located p_x, and send over a *context retrieval request* q_y for retrieving of i_x from p_x.

3.1.2 Context Discovery Platform

A context discovery platform, C, facilitates two operations: *context publishing* and *context lookup*. The *context publishing* operation is accomplished in each context provider p_x by executing the embedded function `publish(Node, msgx)`, where attribute `Node` is one or more network entities to which the published message `msgx` is sent. Attribute `msgx` is the submitted information, which mainly contains ad_x. In some implementation where C caches context information aggregated from the providers in `Node`, `msgx` may carry the updated context information I_x as well.

On the other hand, the *context lookup* operation is the execution of the embedded function `lookup(Node, msgy)`, where attribute `Node`, similar to the function `publish`, is one or more network entities that handle the context lookup in C, and attribute `msgy` is the lookup message submitted to `Node`. `msgy` contains mainly the context lookup

query l_y, and optionally may contain the context retrieval query q_y if `Node` is able to request for context retrieval on behalf of the requester r_y.

Based on the context discovery model in Figure 5, there are different variations of ***C*** in terms of its architecture. We describe the *centralized model* and *broadcast-based model* here, and introduce the *hybrid centralized-decentralized model* adopted in Orion.

3.1.2.1 Centralized Model

C can be a centralized directory where there is a sole database, ***d***, that maintains a set of registered advertisement ***AD***, where $AD \subseteq \bigcup_{x=1}^{m} ad_x$, and all matchmaking processes are taking place in the centralized server (Figure 6). The yellow-page and white-page *context discoverer* in Context Toolkit [19] is one such kind of centralized platform architecture. In the centralized architecture, attribute `Node` in both `publish` and `lookup` function is featuring the same centralized server, and both the `msgx` and `msgy` are headed to the ***d***. Figure 6a is a variation of the centralized server where the context retrieval query q_y can be forwarded to the located context provider p_x by ***d***. In some implementations, `msgx` carries the context information i_x, so that it can be cached by ***d*** and later retrieved by the context requester (Figure 6b). While centralized approach enjoys great query response performance and easy management, it suffers from reliability (e.g. single point of failure) and scalability (e.g. inefficient in handling large query load at the same time, huge memory space for advertisement registration, etc) issues which are undesirable for inter-space scale context discovery.

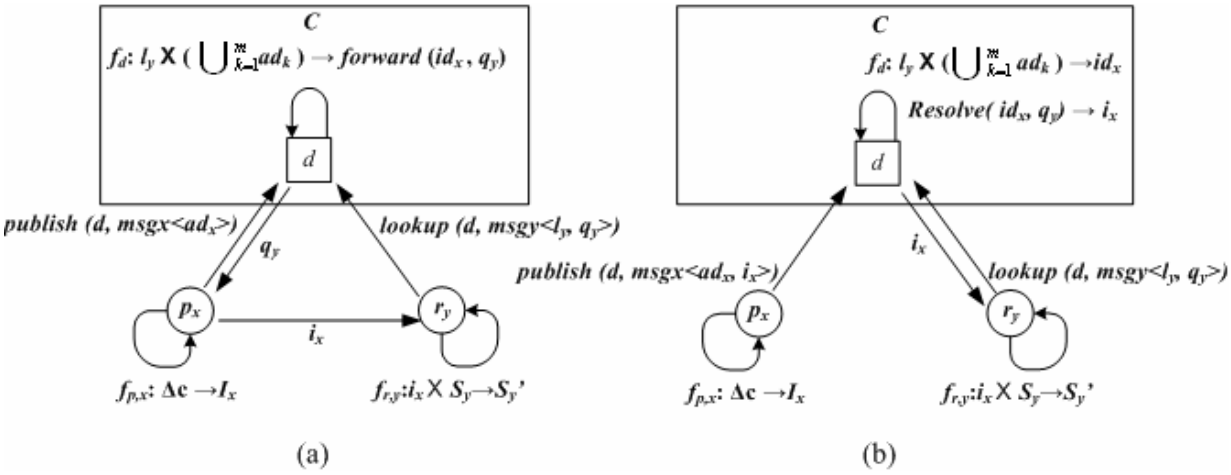


Figure 6. Context discovery model with centralized server; (a) Without context caching; (b) With context caching

3.1.2.2 Broadcast-based Model

On the other extreme, a broadcast-based architecture of ***C*** may exist. There is no special network entities that handle the `publish` and `lookup` operations, but the provider and requester themselves are responsible for it. As a result, attribute `Node` in both `publish` and `lookup` function is the broadcast address. $\boldsymbol{ad_x}$ is broadcasted at periodic interval to notify any listening context requester regarding the existence of $\boldsymbol{I_x}$; while $\boldsymbol{l_y}$ is broadcasted as well to let the relevant context provider who is listening to the broadcast channel to indicate that a matching request is located (see Figure 7).

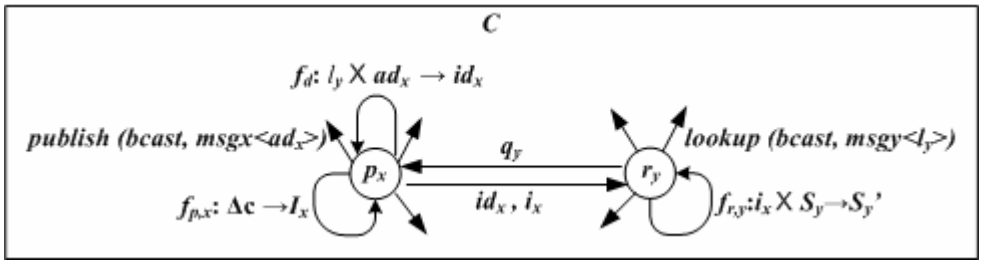


Figure 7. Broadcast-based context discovery model

The context discovery function f_d is therefore carried out by all context providers independently. Universal Plug n Play (UPnP) and the Bluetooth Service Discovery Protocol are adopting such fully decentralized architecture. While broadcast-based approach avoids the single point of failure and performance bottleneck at a centralized location, the broadcast range can be limited within an enclosed network boundary. This, again, is not ideal for handling inter-space context discovery.

3.1.2.3 Hybrid Centralized-Decentralized Model

We propose a hybrid centralized-decentralized context discovery model (see Figure 8) which is suitable for inter-space context discovery. In every smart space, a centralized directory service is deployed, such that context publishing and lookup query take place at this centralized directory. To scale context discovery across many smart spaces, these centralized directory services from different smart spaces are linked together forming the service overlay network. Specifically, Orion employs P2P-based decentralized architecture to connect the distributed directory services, where each directory is a super-peer [47] to the set of context resources peers (i.e. both context requester and provider) in a smart space. As a result, the attribute `Node` in `publish` and `lookup` function denotes the directory service associated to the smart space in which the context resources are located in. At each directory service, the set of registered advertisement AD', where $AD' \subseteq AD$, is published by the context providers that reside in the local smart space. Upon receiving a lookup query l_y that cannot be resolved locally, l_y is appropriately forwarded via the service overlay network to the remote directory service where the requested context is registered with. Context retrieval query q_y is carried along in `msgy`, and is forwarded to the located context provider p_x directly upon the successful matching in the directory service in the remote smart

space. Finally, the service overlay network also serves to propagate the retrieved context information i_x back to the context requester.

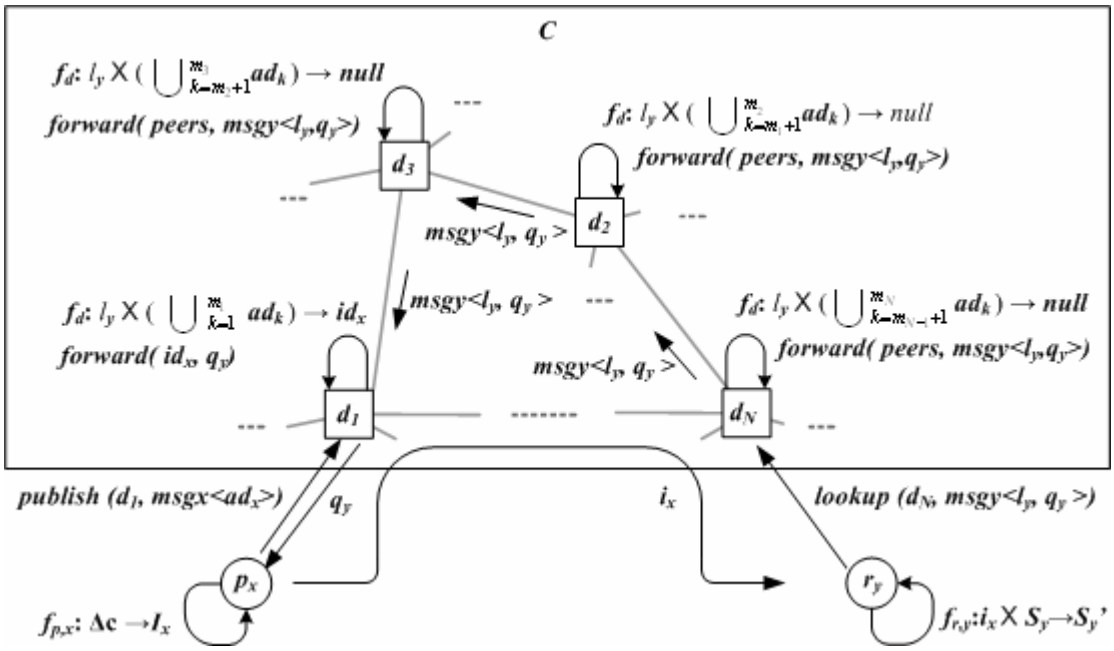


Figure 8. P2P-based centralized-decentralized context discovery model (adopted in Orion architecture)

3.2 Platform Requirements

An inter-space context discovery platform has to be dynamic and scalable to deal with the ubiquitous nature of context providers and context requesters. The research challenges outlined in Chapter 1 result in several requirements that a context discovery platform needs to adhere to. The paragraphs below summarize the various requirements when designing Orion:

- Context discovery platform has to accommodate the ubiquitous nature of both context providers and requesters, which may dynamically join and leave the platform at unpredicted time.

- The platform has to be scalable to handle large number of sensors, devices, query loads, data volumes and users.
- The infrastructure has to enable devices and users to initiate a discovery for specific context regardless of their current space locality.
- Query response time should fall within reasonable range up to user expectation.
- Representation model needs to be expressive and flexible to deal with data heterogeneity.

3.3 Orion Architecture Overview

Orion is a peer-to-peer (P2P) network infrastructure that facilitates context discovery in the context-rich smart spaces. A context provider may publish a context advertisement to Orion for announcing about its existence and about the context information it provides. A context requester, on the other hand, can query Orion to look up for context provider that is able to provide the desired context information. A great emphasis of the Orion architecture is its ability to scale context discovery across many smart spaces, which was highlighted in Chapter 1 as a missing element in current context-aware computing research.

Orion architecture centers on a set of distributed Discovery Gateway (DG). A DG serves the context discovery operations in a smart space. Each DG maintains a set of context advertisements published by context providers in the associated smart space. The DGs from different smart spaces are peers to each other, forming a P2P-based message routing overlay network. As a result, lookup queries for context in remote

smart spaces can be appropriately forwarded to the relevant DG via the overlay network, and thus enable inter-space context discovery to take place.

To interoperate with the heterogeneous context resources, ontology modeling and reasoning technologies from the Semantic Web [51] are employed in Orion. All information is represented with semantics annotation in ontology. Semantic reasoning and matching technique are applied in the DG to perform the matchmaking of the context resources.

3.3.1 Peer-to-Peer Consideration in Smart Spaces

The number of computing entities in our living environment is growing everyday. These entities include, for example, electronic appliances, computers, mobile handheld devices, sensors, digital equipments, as well as software applications such as Web Services, home monitoring software, and personal digital diary. They are independent to one another, but interconnected via wired or wireless network technologies.

These computing entities, when properly enabled with context information communication capability, form a large pool of context providers and context requesters. Under different circumstances and requirements, each entity can be either a context provider or a context requester. For example, a Conference Assistance application [67] running on a handheld device, such as mobile phone and PDA, not only serves as context requester requesting context information about the conference session (e.g. conference schedule, building location layout, participant's particulars, etc), but also providing context information such as location of user and identity of audience during the presentation. When comparing such communication model and information management features with those in peer-to-peer (P2P) network, striking

similarities can be discovered. We therefore see great opportunities in employing P2P techniques in context-aware computing.

In P2P system, each peer plays an equal role in exchanging information and services to one another. A peer manages its own set of information and services, provides them to other when being approached, and also retrieves others information and services for own use. This can ensure the correctness and real-time updating of information at all time. A P2P infrastructure enables context information to be self-contained and self-managed by distributed computing entities. This prevents the requirement of mass storage space in a centralized device, and avoids the performance bottleneck frequently encountered in the single server architecture. The P2P computing model can also handle gracefully the ad-hoc nature of pervasive computing devices, with minimal overhead incurred in managing the joining and leaving of devices to and from the smart spaces.

We can analyze the computing entities in pervasive computing smart spaces using two parameters: *mobility* and *computing capability*. Along the mobility axis, a *static* entity is one whose physical location must be fixed at all time in order to function normally, due to either power limitation or unwieldy size. At the other end of the mobility spectrum is a *mobile* entity, which can conveniently roam from one space to another and still remain in operation. Along the computing capability axis, a *low computing capability* entity is equipped with limited memory space and is often used to handle simple task due to its low computing speed. On the other hand, a *high computing capability* entity has large memory space for data storage in addition to executable code storage, and its computing speed can support the execution of multiple complex tasks in real time. There are, of course, a set of computing entities whose computing

capability falls between the two extremes. Examples based on these analyses are shown in Figure 9.

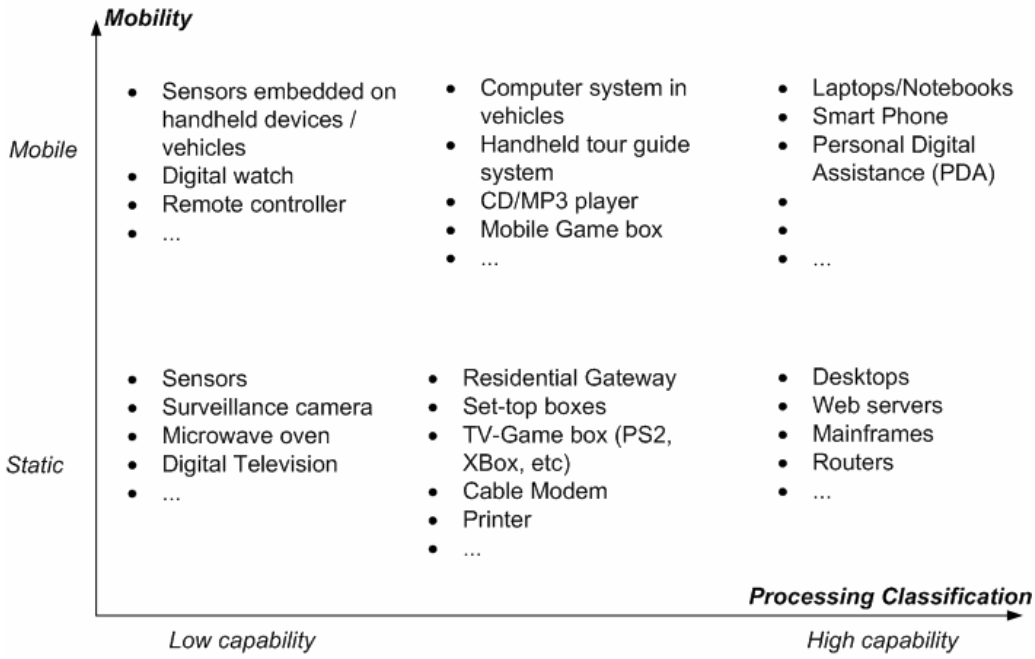


Figure 9. Examples of computing entity peers based on processing capability and mobility classification

To enable communication and interaction between these context resource peers (i.e. both context providers and requesters) distributed in different smart spaces, we adopt a super-peer message routing overlay network [47] approach. A super-peer, elected among the context resource peers in a smart space, operates as a server to a set of client peers. The super-peer in each smart space serves as the gateway of the smart space to communicate with other spaces through an overlay network constructed and maintained among the super-peers. The super-peer overlay network therefore lays the communication infrastructure for the low-end context resource peers (e.g. peers with limited communication range and limited processing power) residing in different smart spaces to communicating with one another. Such approach inherits the self-managing, distributed, low-cost and localized characteristics of a peer-to-peer system, while also

features high manageability and efficiency of a centralized system in a local area. The super-peer overlay network forms the core of the proposed hybrid centralized-decentralized context discovery model. The super-peer in a smart space is known as the Discovery Gateway (DG) in the Orion architecture.

3.3.2 Discovery Gateway

Discovery Gateway (DG) is a super-peer in a smart space that serves as the gateway for the context resource peers in the space to access the Orion context discovery service. The set of DGs from many smart spaces form the message routing overlay network, and they cooperatively provide the following functionalities:

- Registering context advertisement announced by context provider peers residing in the hosting smart space,
- Matching the context lookup query with the advertisement set in a context lookup process,
- Self-organizing into a message routing overlay network in order to provide a context discovery platform that scales across many smart spaces,
- Performing query message routing from one DG to another during the search for remote space context information.

Orion adopts a service-oriented architecture to meet the software engineering challenges. Therefore, the various functionalities in a DG are performed by a cooperative set of service components that operate in the DG architecture. The DG architectural diagram is shown in Figure 10.

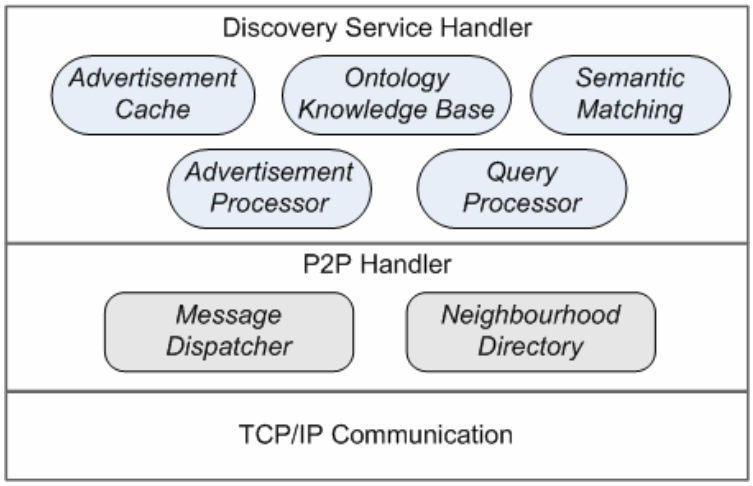


Figure 10. The architectural diagram of a Discovery Gateway

The core service component in a DG is the *Query Processor* that performs the matchmaking of the received lookup query with the advertisement set maintained in the *Advertisement Cache*. It relies on the semantic matching and reasoning algorithm implemented in the *Semantic Matching* component, and refers to the knowledge ontology, such as advertisement template and domain ontology, stored in the *Ontology Knowledge Base*. Context advertisement announced by local context providers are all analyzed and processed by the *Advertisement Processor*, and subsequently stored and maintained in the *Advertisement Cache*.

While the *Discovery Service Handler* layer takes care of the context discovery service, the P2P communication and message routing service are offloaded to the *P2P Handler* layer. The *Message Dispatcher* pre-processes any incoming messages and routes the messages to the relevant service components in the upper layer. For example, lookup queries are transferred to the *Query Processor*, and context advertisement is sent to the *Advertisement Processor*. It also performs the task of message routing in the overlay network, such as forwarding query that cannot be resolved locally to the neighbour peers, and relaying reply message back to the sending DG. The *Neighbourhood*

Directory maintains the list of neighbour peers, mapping their peerID to the respective communication channel. All messages are sent through the TCP/IP communication protocols.

By adopting the component-based development methodology, the DG architecture becomes extensible and flexible. By defining the appropriate service interface, the object-oriented service components can be easily upgraded with different service implementations. Extra functionalities can be introduced by simply installing new service components into the architecture. The layered architecture also ensures loose coupling with the low-level transportation protocol, which is essential for maintaining reliability and connectivity of the overlay network.

The generic architecture of DG has enabled a wide range of computing entity peers to become a candidate DG in a smart space. To become a DG, the peer needs to be equipped with sufficient memory for advertisement storage, and be able to perform semantic matching in the lookup process. On top of that, the candidate peer must be able to establish and maintain overlay communication channel with other DGs, preferably over a long period of time. Judging from these criteria, the computing entity with medium to high processing capability as presented in Figure 9 are all suitable candidate DG in a smart space. To minimize the handover and mobility issue in mobile devices, static-based computing entity is more desirable for its ability to maintain stable overlay links. In current stage, we assume a single DG exists in every smart space, and DG election is not within the current scope of Orion.

3.3.3 P2P-based Overlay Network

All DGs manage local context advertisements relevant to their associated smart space. It is therefore a challenge in Orion to ensure the lookup query searching for context

provider in a remote space can be appropriately and efficiently forwarded to the relevant DG. Orion adopts a decentralized P2P architecture in organizing the DGs, connecting them into a self-organized unstructured peer-to-peer network, called the *Orion Network* (ONet). Each DG peer in ONet maintains its own set of neighbour DGs (i.e. neighbourhood). As a result, ONet provides path connectivity between any two DGs through message relay among the peers. Whenever a sending DG needs to forward a message to a remote DG whose location in ONet is unknown, the message is forwarded to its one-hop neighbours. The forwarding process continues from one DG to another, until finally when the message reaches the destination DG. With ONet, lookup queries can be successfully forwarded to the destined smart space and resolved by the appropriate remote DGs. Figure 11 provides a snapshot of ONet, showing the query message forwarding activities when a smart phone application launches a lookup query to search for sensor information located in a smart space two hops away.

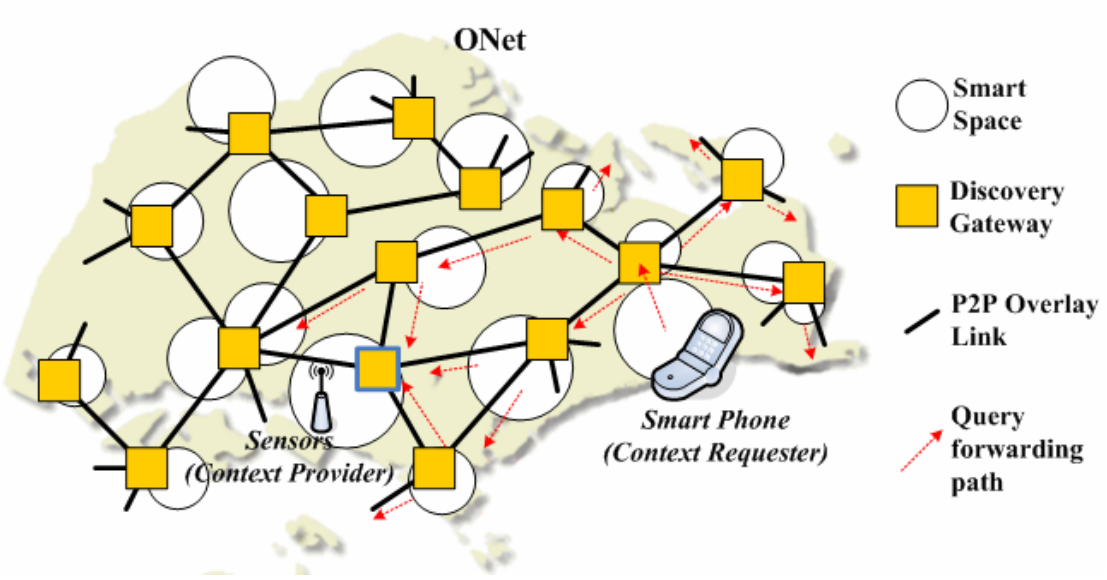


Figure 11. A sensor is discovered by the smart phone application located in another smart space via the Orion Network (ONet)

The unstructured-based ONet is facing similar drawback in Gnutella P2P network – high redundant processing and low message efficiency [41] [46], [68]. In the forwarding process, a query message may be duplicated many times, and a DG may need to process the same query message more than once. To overcome the problems, Orion introduces the *Semantic Community* (SeCOM). A SeCOM is a cluster of DGs that shares similarity in the semantic of context information they are registered with. These DGs form the semantic overlay network, such that lookup queries are routed to and forwarded only within the appropriate SeCOM that is able to resolve them (Figure 12). The size of a SeCOM is fractional compared to the overall size of ONet, and therefore the number of message flooding can be reduced significantly, and so does the over message redundancy in ONet.

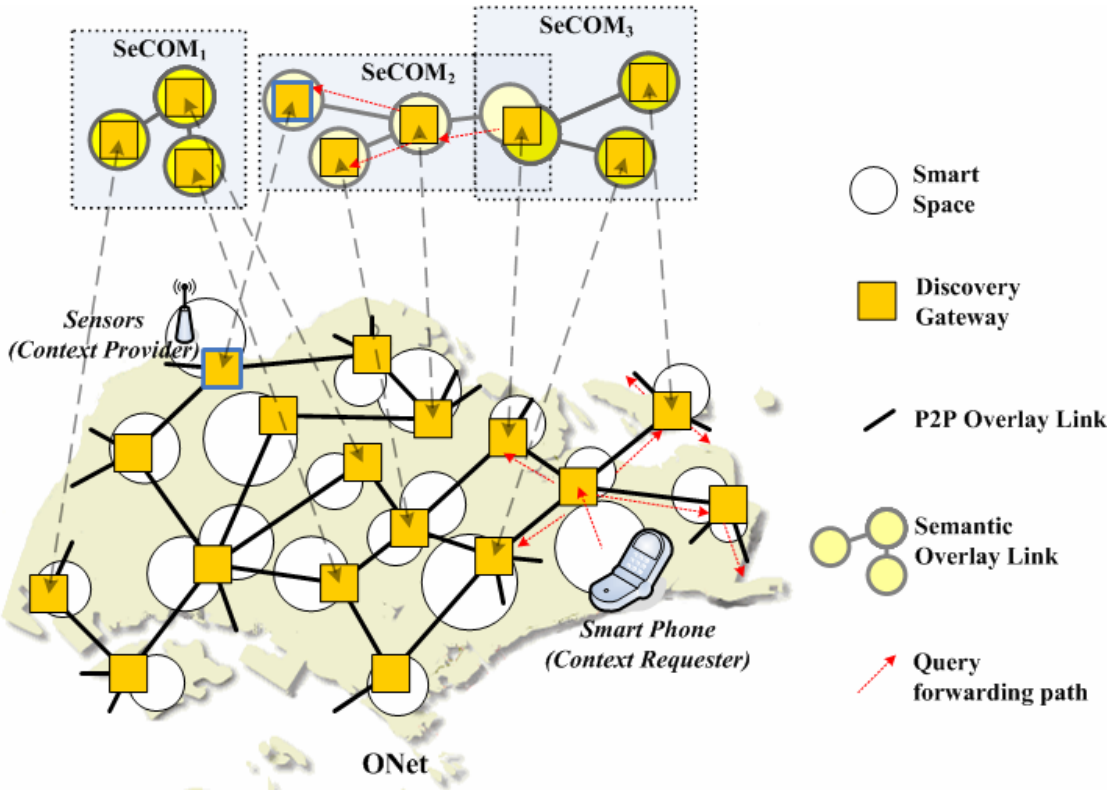


Figure 12. Lookup query is flooded only within the relevant Semantic Community (SeCOM) before reaching the destination DG.

SeCOM exploits geographical location of the context in its clustering criteria. Therefore, DGs registered with context information related to the same geographical location are clustered into the same SeCOM.

Message efficiency reflects the message and process redundancy incurred when forwarding a message to the neighbour peers [41]. Message efficiency drops significantly when the forwarding depth increases. Therefore, we adopt *Iterative Deepening Search* (IDS) [33] approach when flooding message in ONet. IDS ensures the forwarding depth would not go beyond the forwarding depth level where the destination DG is reached. It minimizes the amount of redundant messages, and therefore improves message efficiency of the flooding-based search operation.

Finally, when the destination DG is reached, a "shortcut link" is established between the sending DG and the destination DG, i.e. destination DG becomes the newest neighbour peer of the sending DG. As such, the result is returned within one-hop distance. It also increases the likelihood of finding destination DG within single hop in future search operation [42].

3.3.4 Ontology Modeling and Reasoning

Smart spaces are scattered with ad-hoc and heterogeneous context resources. To facilitate interoperability between the resources, Orion adopts ontological modeling techniques to model the context information. Ontology provides the autonomous context resources with a common semantic understanding of the represented contextual information, even without prior agreements on how they should interoperate. As a result, it promotes easier context information exchange between the context resources, and accurate matchmaking in the discovery process.

Semantic Web technologies [51] are adopted in defining, interpreting and matching the ontological description. We use the Web Ontology Language (OWL) [56] to define the advertisement template, called the Context Advertisement Ontology (CoAO), for composing the context advertisement. Each context advertisement is an instance of the CoAO, describing the published context information in terms of the provider profile, context domain, access information and matching quality. OWL ontology can be viewed as a set of *context triple*. Context triple consists of (`subject`, `predicate`, `object`) that outlines the relationship between a `subject` and an `object` through the relations `predicate`. An advertisement contains one or more triples stored and maintained in the *Advertisement Cache*. For example, context information "*Mobile phone runs out of battery*" is represented using the triple set "(`MPhone`, `hasBattery`, `Battery`), (`Battery`, `hasPowerLevel`, `0`)" or "(`MPhone`, `powerStatus`, `0`)". By using semantic reasoning techniques, the two triple sets are interpreted as equivalence in terms of their semantics (i.e. transitive equivalence).

Ontology modeling provides the ground for semantic matching in the matchmaking process. We use triple-matching techniques, coupled with semantic reasoning, to match the lookup query with advertisement. Context lookup query, built on the SQL-like RDF Data Query Language (RDQL) [69], supports query over semantic model based on matching of the triple patterns. For example, when the context requester is to check the battery level of the user's mobile phone, a RDQL with triple pattern "(`MPhone`, `powerStatus`, `?x`)" is generated. The pattern matches perfectly with the context information described in the previous example, and therefore the value "`?x = 0`" is returned.

3.3.5 Context Discovery Operations in Orion

Context discovery operations include both the context publishing and context lookup. Orion provides the necessary network infrastructure and semantic matching platform to perform the operations.

The context publishing operation takes place whenever a new context provider is newly deployed in a smart space. The context provider p_x invokes the function `publish(`d_{local}`, msgx<`ad_x`>)` to register the context advertisement ad_x with the local space DG d_{local}. After ad_x is stored in the *Advertisement Cache*, the geographical location meta-context semantics of the ad_x is analyzed to identify the SeCOM where the advertisement is associated to. If d_{local} is not a member of the identified SeCOM, the *joinSeCOM* operation is initiated to join as the member of the relevant SeCOM.

In context lookup operation, the context requester r_y executes the function `lookup(`d_{local}`, msgy<`l_y`, `q_y`>)` to query the local space DG d_{local} for the availability of context provider that matches the description in the lookup query l_y. If d_{local} is unable to resolve the query (i.e. a remote space context is queried), l_y is analyzed for its geographical location meta-context semantics for identifying the relevant SeCOM that can resolve the query. Function `forward(`D'`, msgy<`l_y`, `q_y`>)` is called to forward `msgy<`l_y`, `q_y`>` to the set of neighbour DG peers D'. If the forwarding DG is a member of the identified SeCOM, D' will consist only the neighbours in the SeCOM. Otherwise, D' will include all one-hop neighbours in the ONet. When `msgy<`l_y`, `q_y`>` reaches the destination DG d_{dest}, the lookup query l_y is able to be resolved, and retrieval query q_y is forwarded to the located context provider. A shortcut link is established between d_{dest} and d_{local} in order to facilitate the provision of the retrieved context information within single hop latency.

The operations are summarized and presented as an operation flow chart in Figure 13. The details are discussed, analyzed and evaluated in Chapter 4 and Chapter 5.

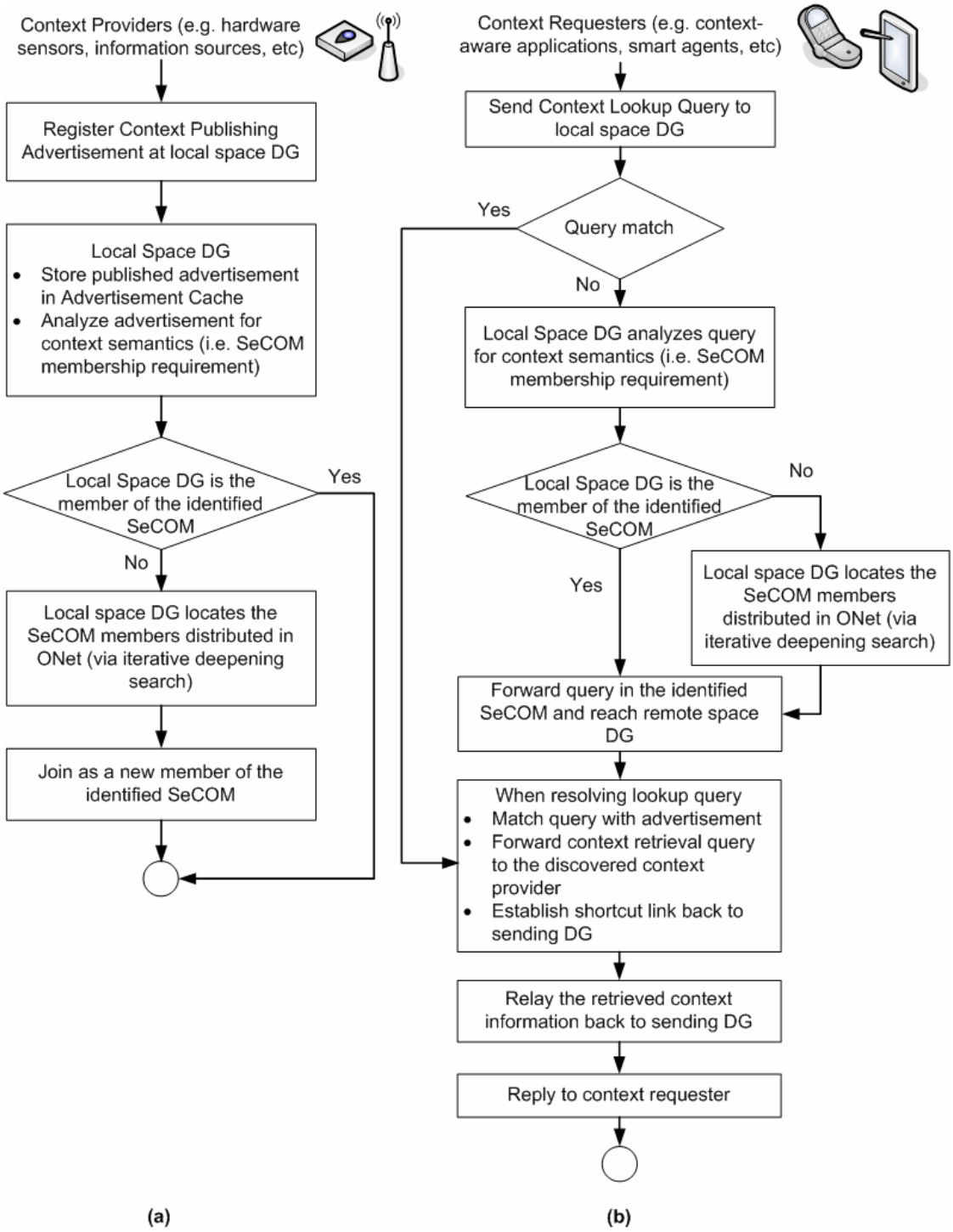


Figure 13. Overview of context discovery operations in Orion. (a) Context publishing, (b) Context Lookup

3.4 Chapter Summary

In this Chapter, the reader is guided through an overview of the Orion architecture. The operation model in Orion is derived from the general context discovery model that involves the context provider, context requester and context discovery platform. We argue that computing entities in pervasive computing fit well into the notion of peer-to-peer computing system, and therefore the Orion architecture is built upon a P2P-based message routing overlay network infrastructure. Specifically, a Discovery Gateway is introduced to function as the super-peer to the resource peers in each smart space, which handles the publishing and lookup of context information, as well as routes message across the overlay network. The context discovery operations supported by Orion are derived, and will be further elaborated and analyzed in Chapter 4 and 5.

CHAPTER 4 P2P NETWORK IN ORION

One of the challenges in inter-space context discovery is to make localized context information discoverable and retrievable across the local smart space boundary. Orion addresses this challenge by using peer-to-peer (P2P) system approach for inter-space and inter-resource communication. In this chapter, the P2P message routing overlay network, called the *Orion Network* (ONet), is introduced. To ensure scalability of the flooding-based search mechanism, several techniques used in unstructured P2P network are modified and incorporated in Orion, which include the RTT probing, Iterative Deepening Search, Semantic Clustering Overlay Network and Shortcut Link. We formalize the search mechanism in Orion, and evaluate the performance via simulations.

4.1 Orion Network (ONet)

Orion Network (ONet) is an unstructured P2P message routing overlay network formed by the set of distributed, autonomous and self-managing Discovery Gateway (DG). DG acts as the super-peer for a set of context provider peers and context requester peers in a smart space, handling advertisement caching, lookup matchmaking and query forwarding across ONet.

We model ONet $\boldsymbol{O}$ as a set of $\boldsymbol{k}$ DGs, $\boldsymbol{D}$, such that $\boldsymbol{O} = \{\boldsymbol{d_i} \mid \boldsymbol{d_i} \in \boldsymbol{D}$, for $\boldsymbol{i} = 1, 2, \ldots, \boldsymbol{k}\}$. A DG $\boldsymbol{d_i}$ manages its own set of $\boldsymbol{k_i}$ neighbour DGs $\boldsymbol{D_i'}$, where $\boldsymbol{D_i'} \subseteq (\boldsymbol{D} \setminus \{\boldsymbol{d_i}\})$, and forms the neighbourhood $\boldsymbol{N_i}$, where $\boldsymbol{N_i} = \{\boldsymbol{d_{i,j}} \mid \boldsymbol{d_{i,j}} \in \boldsymbol{D_i'}$, for $\boldsymbol{j} = 1, 2, \ldots, \boldsymbol{k_i}, \boldsymbol{k_i} < \boldsymbol{k}\}$, such that $\exists$ $link_o$ $(\boldsymbol{d_i}, \boldsymbol{d_{i,j}})$. $link_o$ $(\boldsymbol{d_i}, \boldsymbol{d_{i,j}})$ is the overlay link that connects $\boldsymbol{d_i}$ to its neighbour

$d_{i,j}$, and the link is a bidirectional TCP/IP communication channel, i.e. $link_o\ (d_i, d_{i,j}) = link_o\ (d_{i,j}, d_i)$.

Two assumptions are made when establishing an ONet. First, $\forall i$, we assume $N_i \neq \phi$. This means there is no isolated d_i. The second assumption supposes that O is a fully connected graph network with all DGs connected to one another via at least one message forwarding path.

4.1.1 Bootstrapping ONet

Bootstrapping process enables a newly emerged d_i to establish its N_i. d_i obtains an initial list of $d_{i,j}$ through the broadcasting mechanism, an approach similar to the Gnutella[14] peer-to-peer system. A *ping* message is broadcasted, and several *pong* reply messages are received from a set of potential neighbour DGs who have heard the *ping* message. The ping and pong message carry the timestamp[15] information indicating the time the message is generated. As a result, based on the replied *pong* messages, the round trip time (RTT) for each communication channel connected to the potential neighbours is calculated and sorted. We employ RTT as a simple but realistic measurement metric in an attempt to ensure topology matching between ONet overlay links and underlying physical network [46]. Based on the list of potential neighbour DGs sorted according to the measured RTT, k_i neighbour DGs (where $k_i << k$) whose communication RTT are the shortest, are selected to form the neighbourhood N_i.

A *Neighbourhood Directory* (NDir) is maintained in d_i to manage the 1-hop neighbourhood information. NDir is a collection of 3-tuple <*IP address*, *port number*, *RTT*>. Each entry in the NDir maps the communication channel to neighbour $d_{i,j}$, i.e.

[14] http://www.gnutella.com

[15] The clock in DGs can be synchronized by different well-known techniques, such as using the Network Time Protocol (http://www.ntp.org).

$link_o$ $(\boldsymbol{d_i}, \boldsymbol{d_{i,j}})$, with the IP address and port number, as well as message propagation RTT for the channel.

4.1.2 Leaving ONet

DGs in $\boldsymbol{O}$ conduct RTT-probing to all neighbours in every time period $\boldsymbol{t_p}$. Other than to update the RTT for each communication link recorded in the NDir, it could also response to the unexpected leave of a neighbour DG from Orion. Unexpected leave can be caused by the sudden failure of a DG, or network connection problem. We conclude a neighbour DG $\boldsymbol{d_{i,j}}$ has left ONet unexpectedly when no reply to RTT–probing is received after 3 $\boldsymbol{t_p}$ periods. This results in the removal of $link_o$ $(\boldsymbol{d_i}, \boldsymbol{d_{i,j}})$.

On the other hand, if $\boldsymbol{d_i}$ leaves $\boldsymbol{O}$ expectedly (e.g. when no other context resources remain in the smart space), $\boldsymbol{d_i}$ will terminate all $link_o$ $(\boldsymbol{d_i}, \boldsymbol{d_{i,j}})$ by sending a *bye* message to all neighbours in $\boldsymbol{N_i}$.

4.1.3 Search in ONet

ONet is an unstructured P2P network. A message is forwarded to a destination DG by relaying the message from one DG to another until the destination is reached, or until TTL (time-to-live) of the message expires. As highlighted in Chapter 2, such flooding-based search mechanism causes a great amount of redundant messages and redundant processings at each intermediate relay node. Furthermore, message redundancy increases significantly for each hop increment in the message forwarding [41].

To minimize message redundancy in ONet, the *Iterative Deepening Search* (IDS) is adopted in Orion. IDS is a well known technique in the field of Artificial Intelligence for searching over state space [70], while [30] and [33] apply it for searching in a P2P file sharing system.

IDS performs successive breadth-first search with increasingly larger *depth range*. A *depth range* is a logical boundary that encapsulates the DGs to be reached during a message forwarding process in one search iteration. When a DG (i.e. the query initiating DG) initiates a query, the query message is only forwarded for $\boldsymbol{h}$ hops. Those DGs that receive the message falls within the level 1 depth range. If the requested resource is not located by DGs in level 1 depth range, search iteration 2 begins at level 2 depth range by further forwarding the query for another $\boldsymbol{h}$ hops. The expanding of depth range continues until the search reaches the destination DG, or when it reaches the maximum depth level. Figure 14 shows the example of two depth ranges when the query initiating DG performs IDS. With IDS, we can ensure that when the destination DG is located in depth range of depth level γ, DGs in depth range beyond depth level γ do not need to process and forward the query.

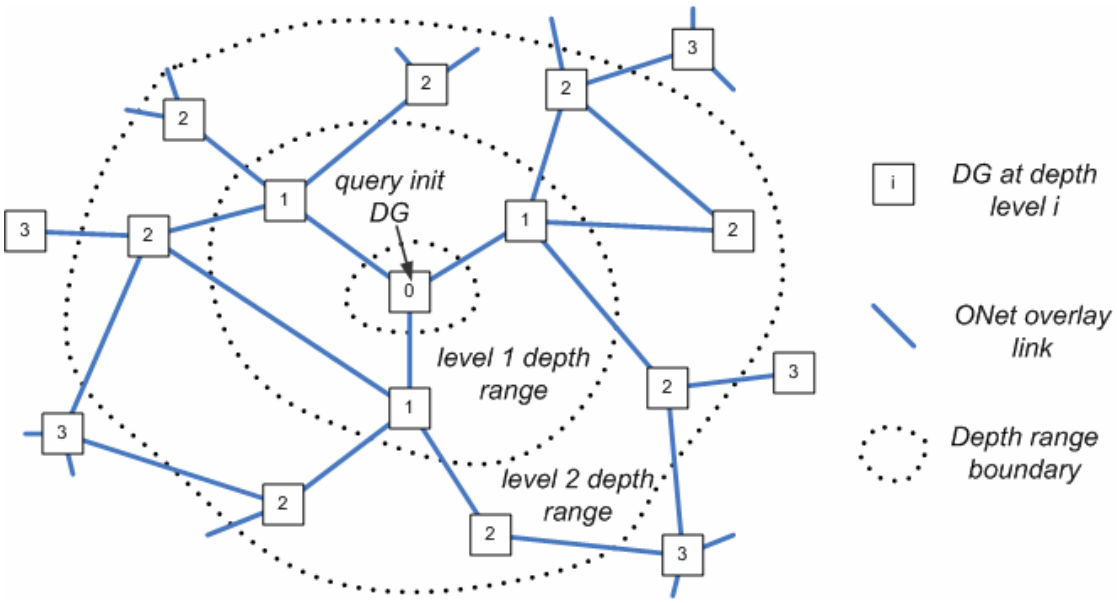


Figure 14. Node coverage at different depth range under the Iterative Deepening Search mechanism (with h = 1).

To execute IDS, each DG in ONet adheres to a *deepening policy* ρ, where $\rho = \{\boldsymbol{h}, \boldsymbol{\omega_{max}}, \boldsymbol{t_\gamma}\}$. $\boldsymbol{h}$ is the number of hops a message is forwarded in every search

iteration (i.e. searching depth in each depth level), ω_{max} is the maximum number of search iteration to be executed, and t_γ is the basic waiting time between successive iterations. The DGs at the range boundary of depth level γ caches a query message for a period T_γ, where $T_\gamma = \gamma \times t_\gamma$. If the S*topSearch* message from the query initiating DG is not received within T_γ, the query message forwarding will be continued in the level γ +1 depth range.

Algorithm 1. Initiating the *IDS* by a query initiating DG d_{init} who wants to search for the resource **φ**.

```
1:  Input: query for resource Q_φ, NeighbourList
2:  Output: located resource φ
3:  Procedure: Init_IDS
4:  Begin
5:  search depth per iteration h ← deepening policy ρ
6:  basic wait time per iteration t_γ ← deepening policy ρ
7:  search depth level γ ← 1
8:  Broadcast (NeighbourList, Q_φ , h, γ)
9:  For each γ < ω_max ← deepening policy ρ
10:       Wait (γ x t_γ)
11:       If resource φ found then
12:            send StopSearch to DGs with h = 0
13:            Break
14:       Else
15:            If receive duplicate Q_φ then
16:                 Discard(Q_φ )
17:            depth level γ ← γ + 1
18:            Send ContinueSearch to DGs with h = 0
19: Wait ( γ x t_γ )
20: Return φ
21: End.
```

Algorithm 1 formalizes the execution order in the DG that initiates the IDS (i.e. the initiating DG, d_{init}, where $d_{init} \in D$). The procedure relies on two parameters: Q_φ is the query for the resource **φ** (e.g. lookup query, join SeCOM query, etc), and *NeighbourList* is the set of one-hop neighbour DGs. In the first search iteration, the DGs in *NeighbourList* are broadcasted with Q_φ, together with the IDS parameters h and γ (line 8). d_{init} waits for a period of ($\gamma \times t_\gamma$) before proceeding to the next step (line

10). When **φ** is not found in this search iteration, the DGs at the current depth range boundary are notified with a *ContinueSearch* message. It signals to them to carry on the search with an incremented depth level (line 15-18). If $\boldsymbol{d_{init}}$ is replied with the discovered **φ**, the search will end at the current iteration, and *StopSearch* message will be sent to DGs at the depth range boundary (line 11-13). **φ** is returned as a result (line 21). For resource **φ** that does not exist in any smart spaces, the search ends after $\boldsymbol{\omega_{max}}$ number of iterations are completed.

Algorithm 2. Performing the *IDS* by relay DGs $\boldsymbol{d_{relay}}$

```
1:  Input: Q_φ , h, γ, NeighbourList, VisitedNeighbours
2:  Procedure: Perform_IDS
3:  Begin
4:  If h > 0 then
5:      decrement h ← h - 1;
6:      Forward (NeighbourList \ VisitedNeighbours, Q_φ, h, γ)
7:  Else
8:      basic wait time per iteration t_γ ← deepening policy ρ
9.     maximum search iteration ω_max ← deepening policy ρ
10:     Wait (γ x t_γ)
11:     If receive ContinueSearch then
12:         h ← deepening policy ρ
13:         γ ← γ + 1
14:         Forward (NeighbourList \ VisitedNeighbours, Q_φ , h, γ)
15:     Else
16:         If receive StopSearch OR duplicate Q_φ OR γ equals to ω_max then
17:             Discard (Q_φ )
18: End.
```

When a DG is able to resolve $\boldsymbol{Q_{\varphi}}$, the located resource **φ** is returned to $\boldsymbol{d_{init}}$. On the other hand, when $\boldsymbol{Q_{\varphi}}$ cannot be resolved, the function *Perform_IDS* (Algorithm 2) is executed to continue the forwarding of $\boldsymbol{Q_{\varphi}}$. The DG therefore becomes the relay DG, $\boldsymbol{d_{relay}}$, where $\boldsymbol{d_{relay}} \in (\boldsymbol{D} \setminus \{\boldsymbol{d_{init}}\})$. When hop count $\boldsymbol{h}$ is not zero, it indicates that $\boldsymbol{d_{relay}}$ is not a DG at the depth range boundary, and therefore $\boldsymbol{Q_{\varphi}}$ is forwarded to all one-hop neighbour DGs that have not processed the query before (line 4-6). For $\boldsymbol{d_{relay}}$ at the search range boundary (i.e. when $\boldsymbol{h} = 0$), the forwarding process is paused for ($\gamma \times \boldsymbol{t_{\gamma}}$)

period of time (line 10). The reception of *ContinueSearch* message during this waiting period denotes that the searching has not ended, and therefore $\boldsymbol{Q}_{\varphi}$ is forwarded to one-hop neighbour DGs that are yet to process the query (line 11-14). On the other hand, if *StopSearch* message is received, or when the maximum search iteration has reached, the search process will be terminated and $\boldsymbol{Q}_{\varphi}$ is discarded.

4.2 Semantic Community (SeCOM)

Other than Iterative Deepening Search, we aim to reduce the redundancy by limiting the number of DGs in ONet that can involve in the process of relaying a message to the destination DG. This can be achieved effectively by restricting the flooding of message within a sub-set of DGs.

As a result, multiple semantic clustering overlay networks, known as the *Semantic Community* (SeCOM), are formed. SeCOM is formed by grouping a set of DGs that are registered with context information that has identical *membership requirement* features. To join a SeCOM, a DG needs to satisfy the membership requirement of the specific SeCOM. It is extracted from the meta-context of the registered context. With SeCOM, we can forward a lookup query only within the SeCOM whose membership requirement matches the meta-context of the requested context information, and resolve the query by one of the member DG in the SeCOM.

SeCOM with membership requirement $\boldsymbol{m}$ is modeled as a set of $\boldsymbol{k_m}$ member DGs, $\boldsymbol{D_m}$, where $\boldsymbol{D_m} \subseteq \boldsymbol{D}$, such that SeCOM $\boldsymbol{S_m} = \{\boldsymbol{d_{m,j}} \mid \boldsymbol{d_{m,j}} \in \boldsymbol{D_m}$, for $\boldsymbol{j} = 1,2,..., \boldsymbol{k_m}, \boldsymbol{k_m} \leq \boldsymbol{k}\}$. $\boldsymbol{d_{m,j}}$ is a member of SeCOM $\boldsymbol{S_m}$, and $\exists\, \boldsymbol{d_{m,j'}}$ where $\boldsymbol{d_{m,j'}} \in \boldsymbol{D_m}$, $\boldsymbol{j'} = 1,2,..., \boldsymbol{k^m}$, $\boldsymbol{j'} \neq \boldsymbol{j}$, such that $link_m$ ($\boldsymbol{d_{m,j}}$, $\boldsymbol{d_{m,j'}}$) is the semantic overlay link between member $\boldsymbol{d_{m,j}}$ and $\boldsymbol{d_{m,j'}}$, established

based on membership requirement ***m***. Semantic overlay link is bidirectional, therefore $link_m$ ($d_{m,j}$,$d_{m,j'}$) = $link_m$ ($d_{m,j'}$,$d_{m,j}$).

Similar to ONet, we make the following assumption: $\forall$ ***j'***, ***j***, such that ***j'*** $\neq$ ***j***, ***j'***, ***j*** = 1,2,..,k_m, $k_m \leq k$, we assume $path_m$ ($d_{m,j}$, $d_{m,j'}$) always exists, where $path_m$ ($d_{m,j}$, $d_{m,j'}$) is the message forwarding path between $d_{m,j}$ and $d_{m,j'}$ in SeCOM S_m. Therefore, S_m is a fully connected graph network.

Figure 15 presents a snippet of the Orion P2P overlay infrastructure. d_1 to d_6 are the DGs in ONet. Each of them has at least 2 neighbour DGs, and they are connected via the ONet overlay link. On top of that, d_1, d_4 and d_6 are the member of SeCOM with membership ***m1***. Therefore, they form S_{m1} and establish semantic overlay link that is identified by ***m1***. d_1 is also the member of S_{m2}, and a semantic overlay link is set up between d_1 and d_3 who is also a member of S_{m2}.

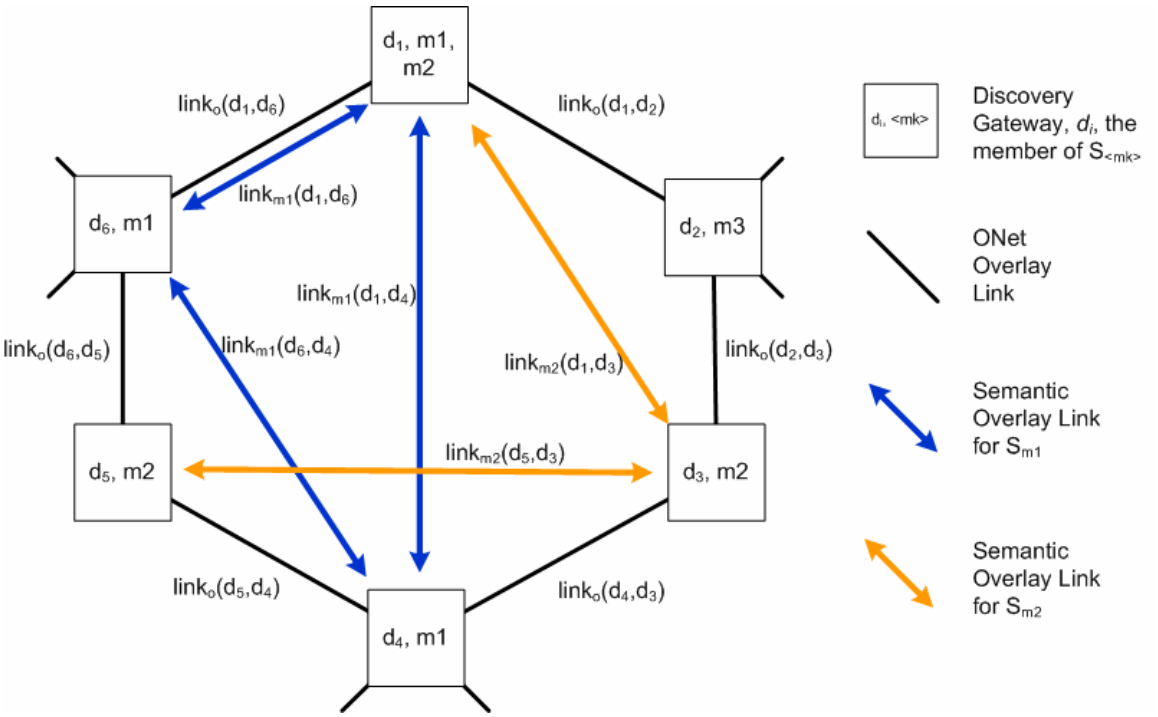


Figure 15. Six DGs in Orion (d_1 to d_6) form their own neighbourhood in ONet and SeCOM, in which the membership requirements include ***m1***, ***m2*** and ***m3***

4.2.1 Meta-context as the Membership Requirement

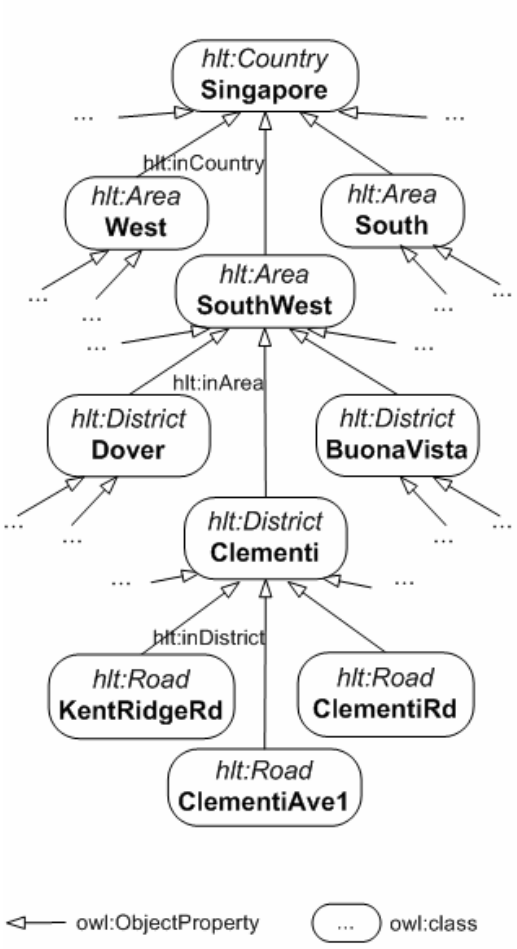
Meta-context represents the metadata features that identify the locality of interest of the context information. For all context information, it contains meta-context that answers these questions - *What is the context about? Where and when does the context take place? Who is the context for? How is the context generated?* For example, the context "*John is in Room 1 at noon*" contains the following meta-context:

- *What*: Indoor location context
- *Where*: Building A (where Room 1 is located)
- *When*: 12 – 1 pm
- *Who*: John
- *How*: Produced by RFID location tracking system

Orion exploits the semantics in the meta-context as the membership requirement $\boldsymbol{m}$ in forming SeCOM $\boldsymbol{S_m}$. Specifically, we observe that a context requester is likely to look up for context related to specific spaces of interest, such as the context in the "been-to" and "going-to" spaces. For that reason, Orion adopts geo-location meta-context as the membership requirement of a SeCOM. Geo-Location meta-context denotes the geographical location of the smart space where the context information is relevant to, i.e. the *Where* factor. Therefore Orion forms a SeCOM by clustering DGs that are registered with context information relevant to certain geographical area.

To begin with, Orion classifies geo-location meta-context based on the proposed *Hierarchical Location Taxonomy* (HLT) defined according to the geographical location of Singapore. HLT defines areas, districts, and roads segment of Singapore in hierarchical order based on its granularity level. Figure 16 provides a fragment of the

classification tree. The classification is modeled using OWL Web Ontology Language [56], such that the hierarchical relationships between the class entities can be semantically represented and interpreted by the intelligence in the computer.




```
<rdf:RDF
  ... >
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Road"/>
  <owl:Class rdf:ID="District"/>
  <owl:Class rdf:ID="Area"/>
  <owl:Class rdf:ID="Country"/>
  <owl:ObjectProperty rdf:ID="inArea">
    <rdfs:domain rdf:resource="#District"/>
    <rdfs:range rdf:resource="#Area"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="hasDistrict"/>
    </owl:inverseOf>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="inCountry">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="hasArea"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Area"/>
    <rdfs:range rdf:resource="#Country"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasArea">
    <rdfs:domain rdf:resource="#Country"/>
    <owl:inverseOf rdf:resource="#inCountry"/>
    <rdfs:range rdf:resource="#Area"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasRoad">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="inDistrict"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#District"/>
    <rdfs:range rdf:resource="#Road"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#inDistrict">
    <rdfs:range rdf:resource="#District"/>
    <rdfs:domain rdf:resource="#Road"/>
    <owl:inverseOf rdf:resource="#hasRoad"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#hasDistrict">
    <owl:inverseOf rdf:resource="#inArea"/>
    <rdfs:range rdf:resource="#District"/>
    <rdfs:domain rdf:resource="#Area"/>
  </owl:ObjectProperty>
</rdf>
```

(b)

Figure 16. Hierarchical Location Taxonomy (HLT) based on geographical location in Singapore. (a) graph representation (b) OWL Ontology definition of HLT

Consequently, the geo-location meta-context in the context information can be extracted by classifying and mapping the context into one or more concept class in the HLT ontology. The classification is performed by two functions implemented in a DG:

classify$_c$ (ad_x, i_x, *HLT)*, and *classify*$_q$ (l_y, q_y, *HLT)*. The *context classification function*, *classify*$_c$ (ad_x, i_x, *HLT)*, classifies the context information i_x and its corresponding context advertisement ad_x that are registered by the context provider p_x. The *query classification function*, *classify*$_q$ (l_y, q_y, *HLT)*, on the other hand, classifies the context lookup query l_y and the context retrieval query q_y that are submitted by the context requester r_y. The result of the classification is the mapping to a concept class in HLT ontology that matches the geo-location meta-context in the presented context information or query. The mapping outcome (i.e. the mapped class name in HLT) is used as the membership requirement m of SeCOM S_m. For example, context "*Meeting is going on in Room3*" would be classified into the "*BuonaVista*" class in HLT, because *Room3* is a meeting room in the *Building A*, which is located at *BuonaVista*. The DG that is registered with this context information will therefore become a member DG in $S_{BuonaVista}$.

4.2.2 Join SeCOM

Orion adopts a conservative strategy for a DG to decide when to join a SeCOM. Whenever a DG is registered with the context advertisement ad_x and the context classification function *classify*$_c$ (ad_x, null, *HLT)* returns $m1$, the DG will decide to join SeCOM S_{m1}, given that it is not a member of S_{m1}. This strategy ensures that whenever query classification function *classify*$_q$ (l_y, r_y, *HLT)* returns $m1$, the context lookup operation is able to locate the appropriate context provider from one of the member DGs of S_{m1}.

To join as the member of the SeCOM S_{m1}, the joining DG d_{join}, where $d_{join} \in (D \setminus D_{m1})$ will require the cooperation from the relay DG d_{relay} and the existing SeCOM member DG d_{mem}, where $d_{relay} \in (D \setminus (\{d_{join}\} \cup D_{m1}))$, $d_{mem} \in D_{m1}$. Algorithm 3 outlines the

events taking place in d_{join} during the SeCOM joining process. The joining process happens in two phases. The first phase (line 4-7) is to locate at least one d_{mem} using the Iterative Deepening Search approach described in Algorithm 1 and 2. IDS results in a list of SeCOM's member DGs being discovered, and their ID is returned in *MemberID_List*. For each located d_{mem} in the list, the request to join as a SeCOM member is sent over, and $link_{m1}(d_{join}, d_{mem})$ is established when a reply is received (line 9-11). The process ensures that d_{join} is accepted as a member of S_{m1} by establishing semantic overlay link to at least one of the existing SeCOM members. The *Secom_directory* that maintains the SeCOM neighbourhood information in d_{join} is updated consequently (line 14). If no member of S_{m1} is found in all ω_{max} search iteration, d_{join} will establish S_{m1} with itself as the sole member in S_{m1}.

Algorithm 3. Initiating the *Join_SeCOM* request by a joining DG d_{join} who wants to join as the member of SeCOM S_{m1}.

```
1:  Input: membership requirement m1
2:  Procedure: Init_Join_SeCOM
3:  Begin
4:  Enumeration MemberID_List ← null
5:  JoinSecom ← Init_Join_Message (m1)
6:  ONet_NeighbourList ← ONet_Directory( )
7:  MemberID_List  ← Init_IDS( JoinSecom, ONet_NeighbourList )
8:  If MemberID_List is not empty then
9:      For each d_mem ∈ MemberID_List
10:         Send_Join_Secom_Request (d_mem)
11:         Establish_Secom_Link (link_m1(d_join, d_mem))
12:         If Secom_Directory is full then
13:             Break
14: Update_Secom_Directory(m1);
15: End.
```

Algorithm 4 is applied for all DGs (i.e. d_{relay} and d_{mem}) that take part in the SeCOM joining process. When a DG who is not a member of S_{m1} receives the *JoinSecom* message, it functions as the relay DG d_{relay} that is responsible for forwarding the *JoinSecom* message. The IDS process specified in Algorithm 2 takes place (line 6-9).

On the other hand, when the *JoinSeCOM* message is received by d_{mem}, the joining request will be processed. The ID of d_{mem} is returned to d_{join} (line 11-14), and followed by establishing a semantic overlay link between the two if d_{join} requests for it (line 15-17). However, if the joining request is not accepted due to limitation in neighbourhood size, the *JoiningSecom* message is forwarded to the neighbours of d_{mem} in SeCOM for further consideration (line 18-20). By establishing a semantic overlay link to d_{mem}, d_{join} is able to participate in the subsequent query forwarding events that take place in S_{m1}.

Algorithm 4. Handling the *Join_SeCOM* request by relay DGs d_{relay} and SeCOM member DGs d_{mem}.

```
1:  Input: JoinSecom, h, γ
2:  Procedure: Handle_Join_SeCOM
3:  Begin
4:  If JoinSecom is not duplicate then
5:      m1 ← Get_Mem_Req (JoinSecom)
6:      If Not_SeCOM_Member (m1) then
7:          ONet_NeighList ← ONet_Directory( )
8:          NonFwdNeighList ← Record_Duplicate_Requesting_DG ( )
9:          Perform_IDS ( JoinSecom, h, γ,ONet_NeighList, NonFwdNeighList)
10:     Else
11:         If Secom_Directory is not full then
12:             d_join ← Get_Joining_DG (JoinSecom)
13:             MemberID ← Get_ID( )
14:             Reply_Joining_DG (d_join, MemberID)
15:             If Receive_Join_Request(d_join) then
16:                 Establish_Secom_Link (link_m1(d_mem, d_join ))
17:                 Update_Secom_Directory(m1)
18:         Else
19:             For each d_sneigh ∈ Secom_NeighList ← Secom_Directory(m1)
20:                 Forward (d_sneigh, JoinSecom, h, γ)
21: Else
22:     Discard (JoinSecom)
23: End.
```

4.2.3 Leave SeCOM

When a current member DG of S_{m1} no longer maintains any context advertisement classified as ***m1***, the DG (i.e. d_{quit}, where $d_{quit} \in D_{m1}$) needs to leave the SeCOM to avoid taking part in any message forwarding events in S_{m1}. Algorithm 5 and 6 are derived for this purpose.

Algorithm 5 is executed in d_{quit}, such that all neighbours in S_{m1} are notified about the leave. The *LeaveSecom* message, together with the S_{m1} neighbour list, is sent to each and every SeCOM neighbour of d_{quit} (line 4-6). When the leave request is acknowledged, the semantic overlay link is disconnected (line 7-8).

Algorithm 5. Launching the *Leave_SeCOM* request by a leaving DG d_{leave} to leave SeCOM S_{m1}.

1: *Input*: membership requirement ***m1***
2: *Procedure*: Request_Leave_SeCOM
3: **Begin**
4: *LeaveSecom* ← Init_Leave_Message (***m1***)
5: **For** each $d_{sneigh} \in$ *Secom_NeighList* ← Secom_Directory(***m1***)
6: Notify_Leave_Secom (d_{sneigh} , *LeaveSecom*, *Secom_NeighList*)
7: Wait_Acknowledgement( )
8: Disconnect ($link_{m1}(d_{quit}, d_{sneigh})$)
9: **End**.

Algorithm 6 is executed by the neighbours of d_{quit} in S_{m1}, i.e. d_{sneigh}, where $d_{sneigh} \in (D_{m1} \setminus \{d_{quit}\})$. When the *LeaveSecom* message is received, the leaving process is acknowledged (line 6) and the semantic overlay link to d_{quit} is disconnected as well (line 7). Line 8-14 is to ensure SeCOM graph connectivity by establishing semantic overlay link between the neighbours of d_{quit} in S_{m1} (line 8-14).

Algorithm 6. Handling the *Leave_SeCOM* request by the SeCOM neighbour DG d_{sneigh} in SeCOM S_{m1}.

```
1:  Input: LeaveSecom, Secom_NeighList from d_quit
2:  Procedure: Handle_Leave_SeCOM
3:  Begin
4:  m1 ← Get_Mem_Req (LeaveSecom)
5:  d_quit ← Get_Leaving_DG(LeaveSecom)
6:  Acknowledge_Leave (d_quit)
7:  Disconnect ( link_m1(d_sneigh, d_quit))
8:  For each  d_mem ∈ Secom_NeighList
9:        If Not_Secom_Neighbour (d_mem ) Then
10:             Send_Join_Secom_Request (d_mem)
11:             Establish_Secom_Link (link_m1(d_sneigh, d_mem))
12:             Update_Secom_Directory(m1)
13:             If Secom_Directory is full then
14:                   Break
15: End.
```

4.3 Supporting Context Discovery Events

With ONet and SeCOM established, the infrastructure is ready to support the two context discovery events: *context publishing event* and *context lookup event*.

4.3.1 Context Publishing Event Support

Context publishing event is the series of actions taken place in Orion in response to the publishing of context advertisement by a context provider. Algorithm 7 outlines the activities in the local space DG that handles the context publishing event. Context provider p_x executes function `publish(`d_{local}`, msgx<`ad_x`>)` for registering a context advertisement ad_x to the local space DG d_{local}, where $d_{local} \in D$. First, d_{local} interprets and inserts the ad_x into local knowledge base (line 5). Then the context classification function analyzes ad_x for its associated meta-context, which results in identifying the relevant membership requirement $m1$ (line 6). If d_{local} is not a member of the identified SeCOM S_{m1}, the function *Init_Join_SeCOM(**m1**)* (Algorithm 3) is invoked for d_{local}.

Algorithm 7. Initiating the context publishing event advertised by context provider p_x at the local DG d_{local}

1: *Input*: msgx<ad_x>
2: *Procedure*: Context_Publish
3: **Begin**
4: ad_x ← Get_Advertisement (msgx<ad_x>)
5: *Insert_KB* (ad_x);
6: $m1$ ← $classify_c$ (ad_x, null, *HLT)*
7: **If** (*Not_SeCOM_Member* ($m1$)) **then**
8: Init_Join_SeCOM ($m1$);
9: **End**.

4.3.2 Context Lookup Event Support

The context lookup event is driven by Algorithm 8 and 9. During the context lookup, a context requester r_y first executes function lookup(d_{local}, msgy<l_y, q_y>) to submit the lookup and retrieval query to a local space DG, d_{local}, where $d_{local} \in D$. The *Context_Lookup* function in d_{local} (Algorithm 8) is performed to resolve the lookup query. d_{local} attempts to resolve the context lookup query l_y (line 7), by going through the matchmaking procedure based on local Advertisement Cache (see Chapter 5). When the relevant context provider p_x is available in local space, the context retrieval query q_y is forwarded to p_x to retrieve the updated context information (line 8-10). On the other hand, if l_y cannot be locally resolved, d_{local} will need to search for the appropriate remote space DG (i.e. d_{remote}, where $d_{remote} \in (D_{m1} \setminus \{d_{local}\})$) that is registered with the searching context. Depending on whether d_{local} is a member of S_{m1} ($m1$ being the membership requirement that l_y is classified into), d_{local} initiates the search in either ONet or SeCOM S_{m1}(line 12-17). The *Init_IDS* function derived from Algorithm 1 is invoked to perform the search for d_{remote} (line 17). Once d_{remote} is found, a shortcut link can be established between d_{remote} and d_{local} in order to facilitate the retrieval of context information from the discovered context provider that is resides in the remote smart space (line 18-21).

Algorithm 8. Initiating the context lookup event submitted by context provider p_x at the local space DG d_{local}

1: *Input*: msgy<l_y, q_y>
2: *Output*: the context information i_x
3: *Procedure*: Context_Lookup
4: **Begin**
5: l_y ← Get_Context_Lookup_Query (msgy<l_y, q_y>)
6: q_y ← Get_Context_Retrieval_Query (msgy<l_y, q_y>)
7: id_x ← *Resolve* (l_y);
8: **If** id_x is not null **then**
9: Forward (id_x, q_y)
10: i_x ← Wait_Provider_Reply( )
11: **Else**
12: $m1$ ← $classify_q$ (l_y, q_y, *HLT*);
13: **If** *Not_SeCOM_Member* ($m1$) **then**
14: *NeighbourList* ← ONet_Directory( )
15: **Else**
16: *NeighbourList* ← Secom_Directory($m1$)
17: d_{remote} ← Init_IDS (msgy<l_y, q_y>, *NeighbourList*)
18: **If** d_{remote} is not null **then**
19: **If** $link_o$(d_{local}, d_{remote}) does not exists **then**
20: Create_Shortcut_Link ($link_o$(d_{local}, d_{remote}))
21: i_x ← Receive_Context(d_{remote})
22: **Else**
23: i_x ← null
24: Return i_x
25: **End.**

Algorithm 9 is executed by DGs that perform the IDS message relay (i.e. d_{relay}, where $d_{relay} \in (D \setminus (\{d_{local}\} \cup D_{m1}))$), as well as by the remote space DG that can resolve the query (i.e. d_{remote}). A d_{relay} in ONet would execute line 10-12 to relay the query message to its ONet neighbour based on Algorithm 2. When the query message reaches S_{m1}, the member DG would attempt to resolve the query (line 14). If the query cannot be resolved, IDS is performed within SeCOM (line 22-24). When the query message finally reaches d_{remote}, the matchmaking procedure would have resolved l_y, and q_y is subsequently forwarded to the discovered context provider p_x for retrieval of the updated context information i_x (line 14-17). i_x is sent back to d_{local} via the shortcut link established between them (line 18-20).

Algorithm 9. Performing the context lookup event initiated by local DG d_{local} at the remote DG d_{remote} and the relay DG d_{relay}

```
1:  Input: msgy<l_y, q_y>, h, γ
2:  Output: Reply d_local with the located context information i_x
3:  Procedure: Remote_Context_Lookup
4:  Begin
5:  If msgy<l_y, q_y> is not duplicate then
6:      l_y ← Get_Context_Lookup_Query (msgy<l_y, q_y>)
7:      q_y ← Get_Context_Retrieval_Query (msgy<l_y, q_y>)
8:      m1 ← classify_q (l_y, q_y, HLT)
9:      If (Not_SeCOM_Member(m1)) then
10:         NeighbourList ← ONet_Directory( )
11:         NonFwdNeighList ← Record_Duplicate_Requesting_DG( )
12:         Perform_IDS (msgy<l_y, q_y>, h, γ, NeighbourList, NonFwdNeighList)
13:     Else
14:         id_x ← Resolve (l_y);
15:         If id_x is not null then
16:             Forward (id_x, q_y)
17:             i_x ← Wait_Provider_Reply( )
18:             If link_0(d_remote, d_local) does not exists then
19:                 Create_Shortcut_Link ( link_o(d_remote, d_local))
20:             Forward (d_local, i_x)
21:         Else
22:             NeighbourList ← Secom_Directory(m1)
23:             NonFwdNeighList ← Record_Duplicate_Requesting_DG( )
24:             Perform_IDS (msgy<l_y, q_y>, h, γ, NeighbourList, NonFwdNeighList)
25: Else
26:     Discard (msgy<l_y, q_y>)
27: End.
```

4.4 Evaluation

We analyze the scalability issue of the Orion network infrastructure in terms of its query response efficiency and message communication cost. We would like to evaluate the effect of deploying SeCOM of variable sizes in reducing message redundancy. The analysis is carried out based on the results from simulations.

4.4.1 Evaluation Objectives

In this evaluation, we are interested in answering the following questions:

- What is the query response efficiency under different network sizes (of both ONet and SeCOM) and different operation parameters?
- What is the impact of introducing SeCOM on reducing redundant message processing?
- What is the effect on system performance when the overlay network topology changes?

The simulation results are studied and analyzed in two aspects. The *query response efficiency* shows the hop count required for completing a query under various network conditions and topology settings. The *message communication cost* studies the effect of SeCOM in reducing the redundant processing in each DG and improving the message efficiency. Message efficiency is the ratio of the message coverage and number of forwarded message in each hop. It reflects the message and process redundancy incurred when forwarding a message to the neighbour peers. We compare the results against the performance of Gnutella P2P searching mechanism, one of the most widely adopted techniques in current P2P file sharing applications, such as Kazaa and Bit Torrent.

4.4.2 Simulation Methodology

4.4.2.1 Simulator

The Orion simulation platform was developed using the *Peersim* P2P Simulator[16]. *Peersim* is an open-source component-based Discrete Event Simulator (DES) for simulating P2P network and application. The development framework was written in Java programming language.

[16] http://peersim.sourceforge.net

4.4.2.2 ONet Topology

Based on [43] and [71], a Gnutella-like unstructured-based P2P network exhibits a power-law distribution. Therefore, in the simulation, the ONet topology follows the power law distribution. In Power Law network, most nodes have only a few out links and a tiny number of nodes have a large number of out links, i.e. graph metrics follow the distribution $y \propto x^{\alpha}$ [32].

The *Power-Law Out-Degree Algorithm* (PLOD) [72] is implemented to generate a graph of DG nodes that obeys power-law. In step 1 of PLOD, each DG is assigned a neighbourhood size, based on the distribution $n = \beta x^{-\alpha}$, where n is the neighbourhood size (i.e. node outdegree) and x is a random number picked from uniformly distributed range [1, k] (k is the total number of simulated DGs in ONet). Parameter β and α shape the distribution of average neighbourhood size, where the value of α can influence the mean out-degree of each node, while the value of β can shape the curve of the out-degree exponential distribution [32]. Two ONet topologies are used in the simulation process, and the topology parameters are tabulated in Table 1. As k increases, the values of β and α in each topology are adjusted so that the neighbourhood size distribution remains unchanged despite changes in the network size. The values in Table 1 were obtained through experiment.

The step 2 of PLOD is to establish links between the DGs. First, all DGs with no connected neighbours are placed in the "*unconnected*" set. Subsequently, the DGs with at least one connected neighbour are placed in the "*connected*" set. Two DGs, one from each set, are picked randomly and a link between them is formed if neither reaches the outdegree limit. Eventually, when the "*unconnected*" set is empty, two

DGs from the "*connected*" set are chosen instead. The DGs that reach the outdegree limit are removed from the "*connected*" set. This iterative process ends when all DGs are connected up to the neighbourhood size allocated in step 1. As a result of this iterative process, the established ONet is a fully connected graph which obeys the Power Law.

Table 1. Parameters used in generating the two ONet topologies

Network size (k)	β	α	**Average outdegree (n)**	**Max n (n_{max})**	**Min n (n_{min})**
Topology 1					
10000	40	0.29	3.3283	25	2
25000	34	0.245	3.284	26	2
50000	28	0.208	3.303	26	2
75000	28	0.2	3.305	25	2
100000	26	0.19	3.256	26	2
125000	26	0.185	3.251	24	2
150000	24	0.175	3.258	24	2
Topology 2					
10000	30	0.2	5.278	23	4
25000	28	0.175	5.258	24	4
50000	26	0.155	5.267	26	4
75000	26	0.149	5.268	23	4
100000	24	0.138	5.223	24	4
125000	24	0.135	5.245	22	4
150000	24	0.133	5.223	24	4

4.4.2.3 SeCOM Topology

A single SeCOM is established in the simulation process. The number of member DGs in the SeCOM is θ % of the total number of DGs in ONet (i.e. SeCOM size $k_s = k \times \theta\%$). These SeCOM members are randomly chosen among the DGs in ONet. The SeCOM topology is established using step 2 of PLOD, based on the maximum outdegree assigned for each DGs when establishing the ONet. In the simulation, we

use θ=0% to denote an ONet without any SeCOM, which resembles the pure flooding-based unstructured P2P network such as the Gnutella.

4.4.2.4 Simulation Process

The resource searching operation in Orion is simulated under different ONet size k, different SeCOM size k_s, and different ONet topology (i.e. topology 1 and topology 2 under different β and α values (see Table 1)). In every operation, a lookup object is placed in a randomly picked SeCOM node (i.e. the "destination DG"), and a random node in ONet is selected as the query node that would initiate a lookup event (i.e. the "sender DG"). The searching process follows the message forwarding strategies presented in Algorithm 1, 2, 8 and 9. A deepening policy ρ = {1, 10, 3} was used in the simulation, which indicates that a single hop for each depth range, a maximum of 10 search iterations are made, and 3 simulation cycles of waiting time in each iteration. The deepening policy is resetted when the search proceeds with flooding within the SeCOM. Various performance metrics, such as the traversal path of message, message duplication count, new node discovery count, etc, are recorded by all participating nodes and by the query message in each simulation cycle.

In the simulation, we assume fixed P2P network topology, which is a gross simplified assumption. However, if one assumes that the time to complete a search is short compared to the time and frequency of change in the network topology (i.e. node joining and leaving), the results obtained using these settings are still reflective of performance in real systems.

4.4.2.5 Performance Metrics

3 performance metrics are used in the evaluation:

- *Hop count* (h_{total}) is the number of DG-to-DG links a query has to traverse before it reaches the destination DG. Due to the fact that each link is established between neighbour DGs with the shortest RTT, hop count has direct proportional relation with the query response time. Two types of hop count were measured: hop count in ONet (h_{onet}) and hop count in SeCOM (h_{secom}). h_{onet} is the hop count for the query to travel from sender DG to one of the nearest member DG of the SeCOM. h_{secom} is the path length the query takes to traverse SeCOM before it reaches the destination DG. Clearly, we have $h_{total} = h_{onet} + h_{secom}$.

- *Message coverage in hop i* is the number of first-time-visiting node that is reached when the query is forwarded from hop (i – 1) to i. Message coverage in hop 1 is therefore equal to neighbourhood size n of the sender DG. At any other relay nodes, message coverage is less than or equal to (n - 1).

- *Message count in hop i* is the total number of message being duplicated when the message is forwarded from hop (i – 1) to i.

- *Visited node count* is the total number of nodes that has been involved in forwarding the query message at least once before it reaches the destination DG.

4.4.4 Result Analysis

4.4.4.1 Query Response Efficiency

Query response is a measure of the efficiency with which Orion is able to resolve a lookup query. It is defined to be the time taken for a query to be resolved and sent

back to the requester. In the simulation, the hop count that a query takes to reach the destination DG (h_{total}) reflects the query response efficiency of Orion.

Figure 17 and Figure 18 shows the query response in terms of h_{total} when operating in topology 1 and topology 2 respectively. The flooding-based search mechanism results in the linear increment of h_{total} as the network size k increases. It is obvious that h_{total} in topology 1 is generally larger than that in topology 2. This is because the average outdegree of each DG in topology 1 is small in average when compared to DGs in topology 2. Consequently, in order to accommodate the growth in network diameter, the hop count increases. Similarly, this also explains the much lower linear increment rate in topology 2 compared to topology 1.

When varying the SeCOM size k_s, it is observed that the smaller the θ, the shorter the h_{total}. With a small SeCOM size of about θ=1% in topology 1, h_{total} is shorter by 8% to 27% when k increases from 10000 to 150000. The reduction in hop count, however, is only 6% -14% for topology 2 under similar condition. As k_s increases, the hop count reduction becomes minimal, especially for topology 2.

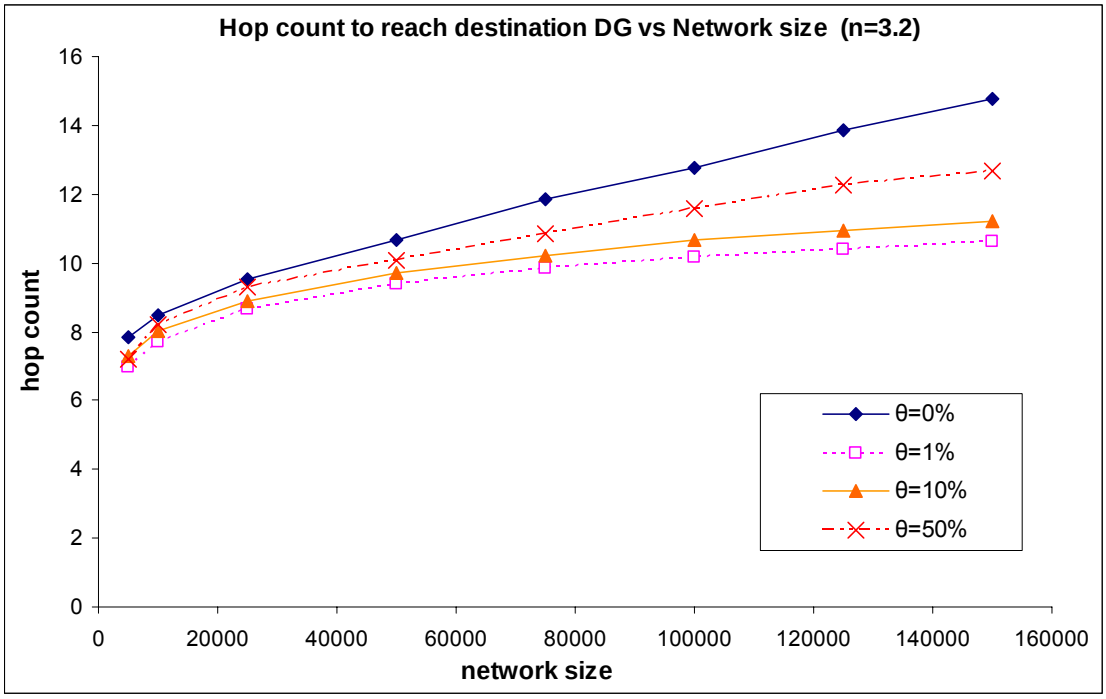


Figure 17. Query response (hop count to reach destination DG) in topology 1

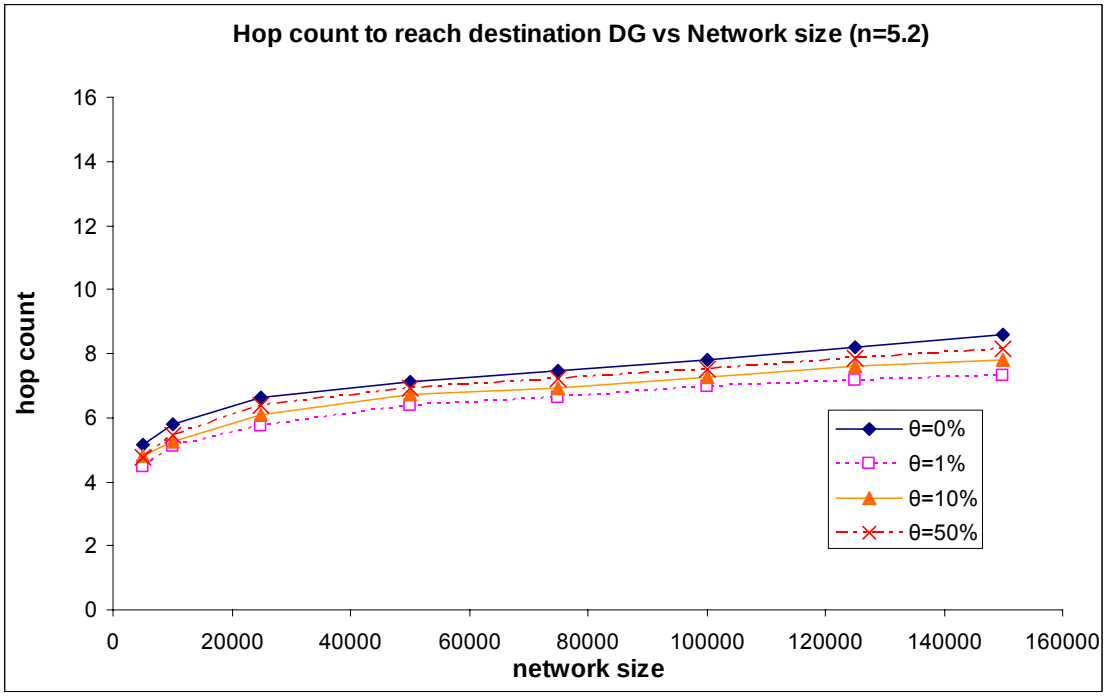


Figure 18. Query response (hop count to reach destination DG) in topology 2

We further analyze h_{total} by separating the hop count into h_{onet} and h_{secom}, i.e. hop count when the query has to travel from sender DG to one of the nearest SeCOM's member

DGs, and hop count when the query traverses SeCOM to reach the destination DGs respectively. It can be observed from Figure 19, Figure 20 and Figure 21 that as θ rises from 1% to 50%, h_{onet} reduces while h_{secom} increases. This phenomenon applies to both topology settings. This is especially true when θ is at 50% value, during which a great portion of h_{total} is actually contributed by the traversal in SeCOM itself.

Interestingly, the value of h_{onet} is small in general for all the three SeCOM sizes. Since SeCOM is just another ONet-like overlay network with less number of nodes, the values of h_{secom} at any θ value are also less than h_{total} of ONet at θ = 0%. The cumulative effect results in the reduction of h_{total} when SeCOM is deployed.

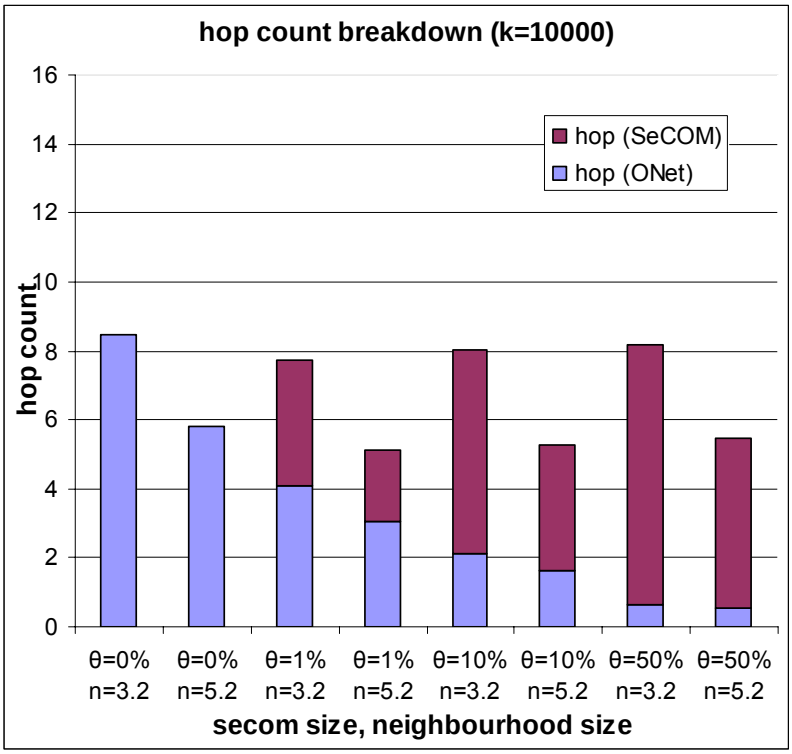


Figure 19. Hop count breakdown analysis for k = 10000

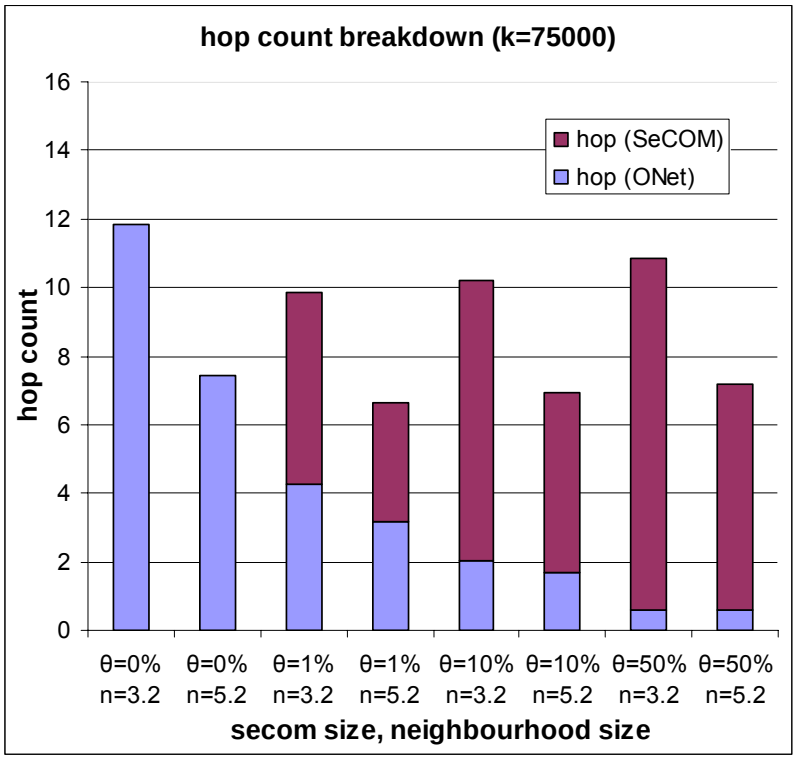


Figure 20. Hop count breakdown analysis for k = 75000

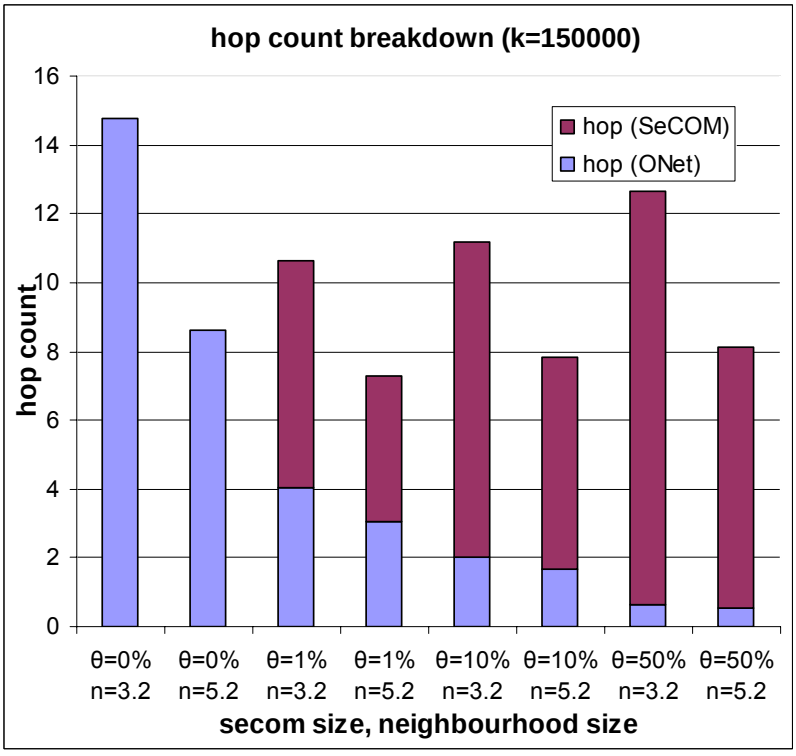


Figure 21. Hop count breakdown analysis for k=150000

4.4.4.2 Message Communication Cost

Message communication cost is the communication overhead incurred as a result of performing search in Orion. It can be measured and analyzed in various dimensions. Here, we only emphasize the effect of SeCOM in reducing message redundancy in ONet.

An unstructured-based P2P network, such as ONet with θ= 0%, will encounter great message redundancy when performing flooding-based search. As shown in Figure 22, in order to forward the query message to the destination DG in a ONet of topology 1 with θ=0%, more than 80% of DG peers are involved in relaying the query message at least once. The number of DG peers involved rises to 90% when the forwarding takes place in ONet of topology 2 with θ=0%(see Figure 23). Furthermore, duplicated messages can visit the same DG for more than once throughout the flooding process.

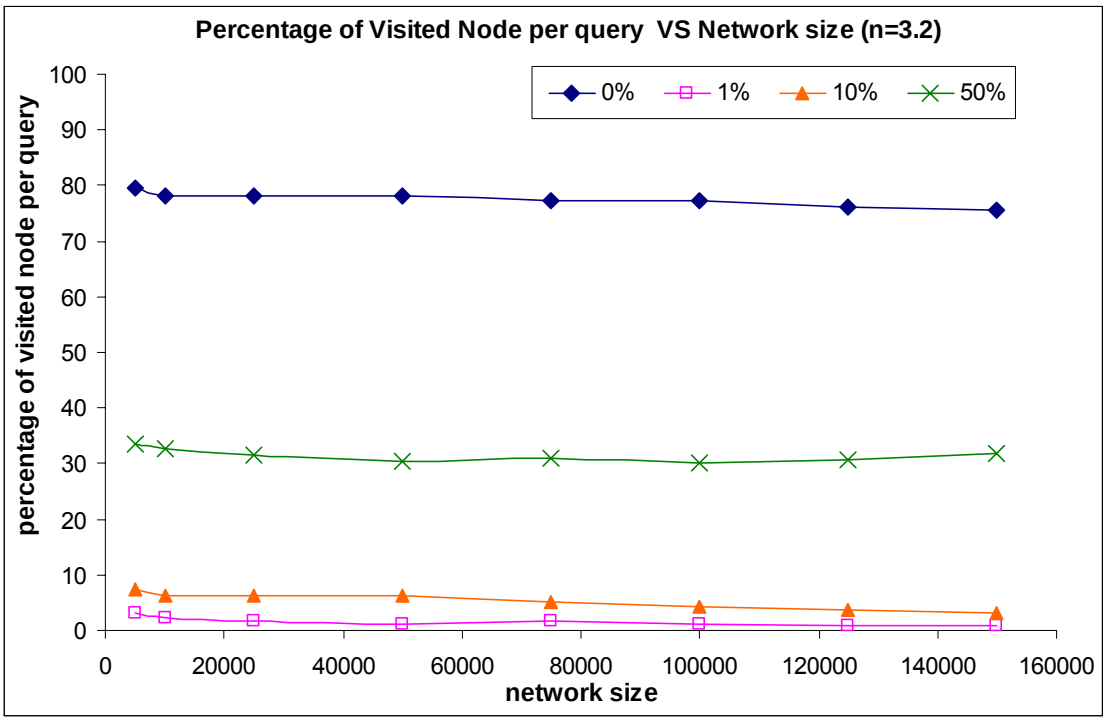


Figure 22. Number of visited nodes per query in topology 1 at θ= 0%, 1%, 10%, 50%

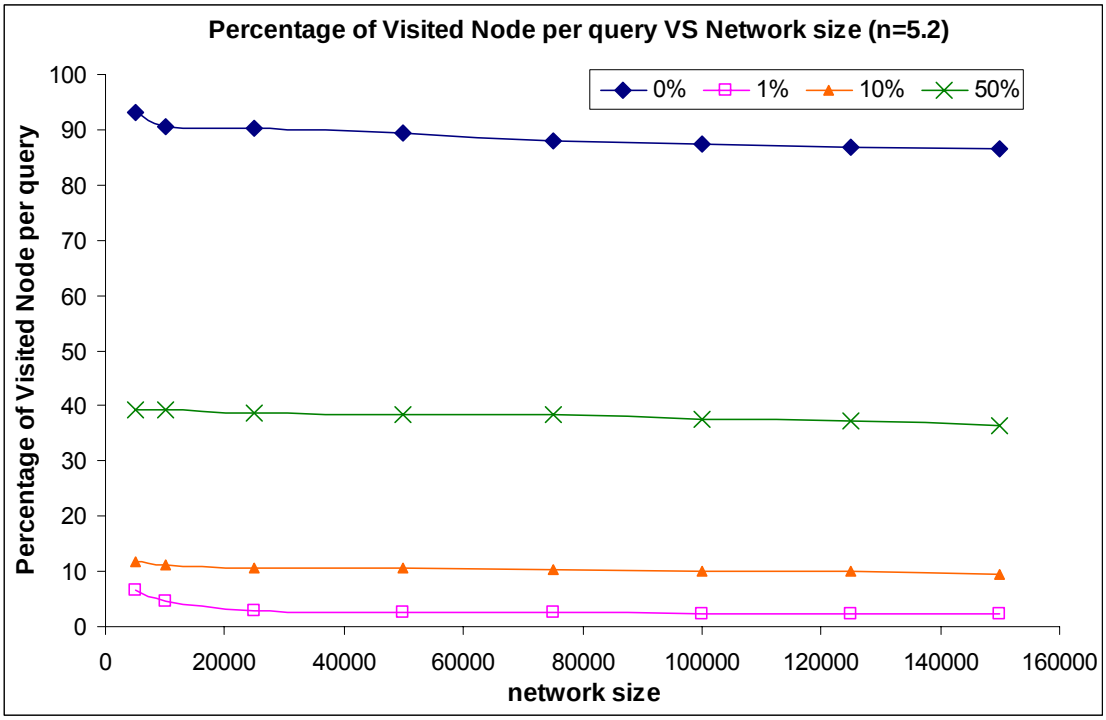


Figure 23. Number of visited nodes per query in topology 2 at θ= 0%, 1%, 10%, 50%

By introducing SeCOM of variable sizes in Orion, we see a sharp decrease in the number of nodes being visited in both topologies. For example, the deployment of a small SeCOM with θ = 1% can reduce the number of visited DGs to a mere 1% and 2% of k for topology 1 and topology 2 respectively. This is because members of the SeCOM can be located with minimal h_{onet} of about 3 to 4 hop counts. Furthermore, the subsequent flooding within the SeCOM only involves the member DGs that sum up to at most 1% of the total populations. As concluded in [41], the majority of redundant messages are produced by nodes that are placed further away (in terms of number of hops) from the sender node. Therefore, by ensuring that the member nodes of a SeCOM can be discovered within several hops, as well ensuring that flooding in ONet does not happen beyond h_{onet} by the use of IDS, the percentage of nodes being visited in the search process can be greatly reduced. The results also show that, for a medium (θ = 10%) and large (θ = 50%) scale SeCOM, the percentage of visited node can

drop to within 8%-11% and 30%-40% respectively. This implies that the smaller the SeCOM size (i.e. small θ value), the greater the reduction in redundant processing.

Furthermore, the message efficiency at various SeCOM sizes is analyzed. Message efficiency is measured as the ratio of the message coverage and number of forwarded message. It reflects the overhead produced by redundant messages in each forwarding hop in the flooding process. The message efficiency at various stages of the flooding process is shown in Figure 24, Figure 25, Figure 26, and Figure 27. Generally, ONet with topology 1 enjoys greater message efficiency in all stages of the flooding process when compared to ONet with topology 2. DGs in ONet with topology 2 have larger outdegree in average. This implies that, whenever a message is to be forwarded to the neighbours, more messages are to be duplicated, and the probability of forwarding the message to a neighbour DG who has been visited before via another path is therefore higher. The cumulative effect is a drop in message efficiency in topology 2.

If we observe the message efficiency rate in SeCOM of various sizes, we can conclude that the smaller the SeCOM size, the better the message efficiency is at different stages of the flooding process. For SeCOM of θ = 1%, message efficiency rate stays higher than 80% for at least the first 65% of the forwarding hop. The query message enters the SeCOM of θ=1% in about hop 3 to hop 4, and the subsequent flooding process in the SeCOM ensures the message efficiency remains high. This explains the sudden spike observed in all four figures for message efficiency in SeCOM of θ=1%. As for SeCOM of θ=10% and θ=50%, the message efficiency at each stage of the flooding process is observed to be higher than that in pure flooding search without SeCOM. However, the extra gain in message efficiency for such medium to large SeCOM is insignificant when compared to the gain obtained in smaller SeCOM of size θ<10%.

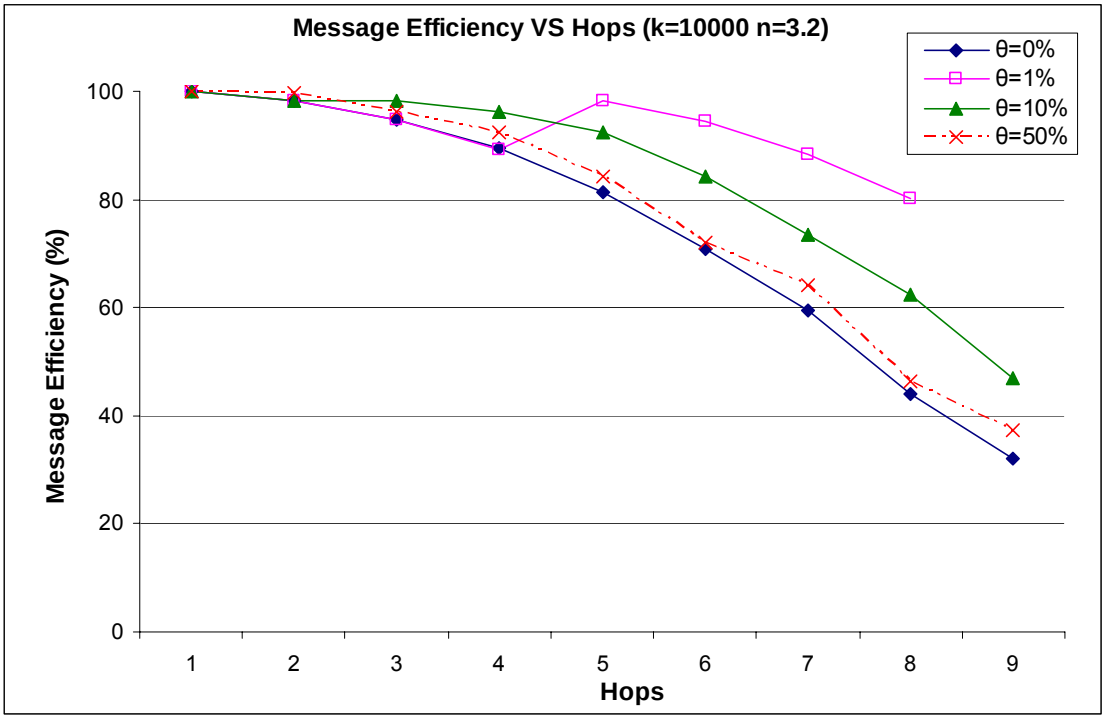


Figure 24. Message Efficiency in topology 1 with k=10000 at various θ values

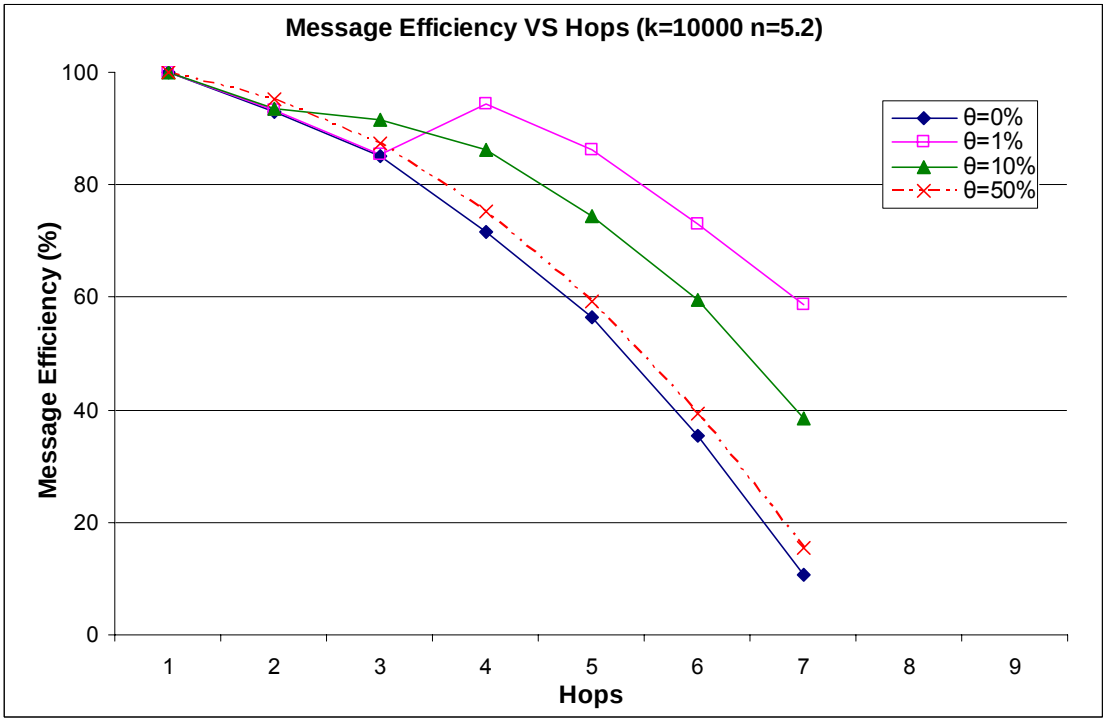


Figure 25. Message Efficiency in topology 2 with k=10000 at various θ values

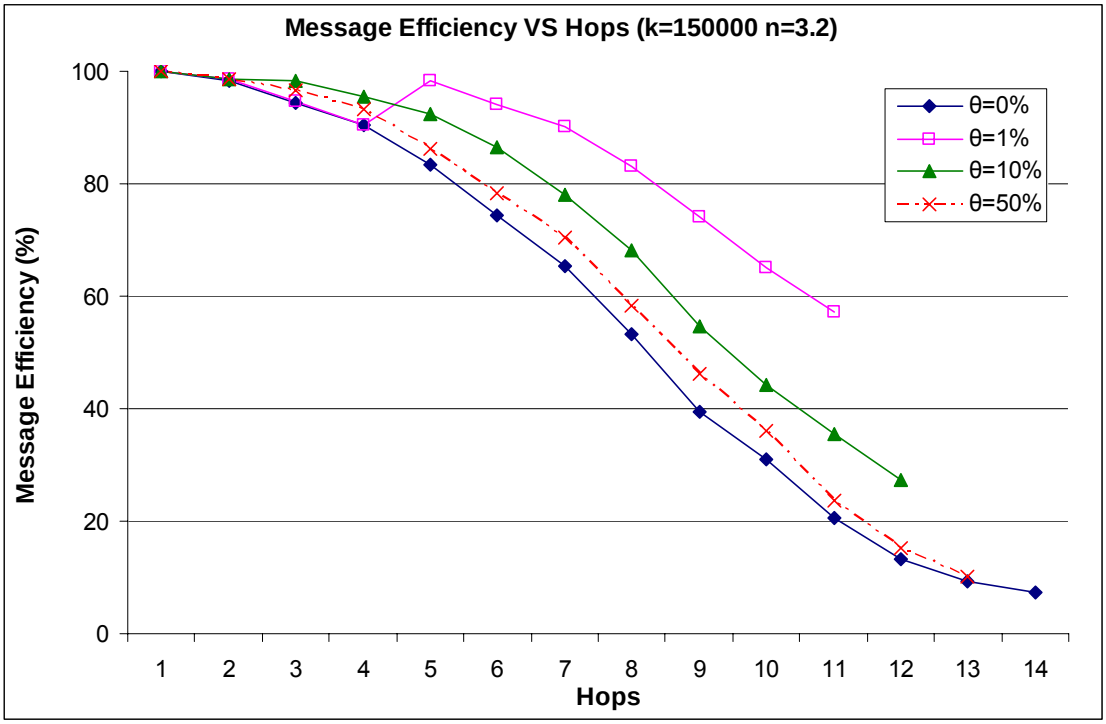


Figure 26. Message Efficiency in topology 1 with k=150000 at various θ values

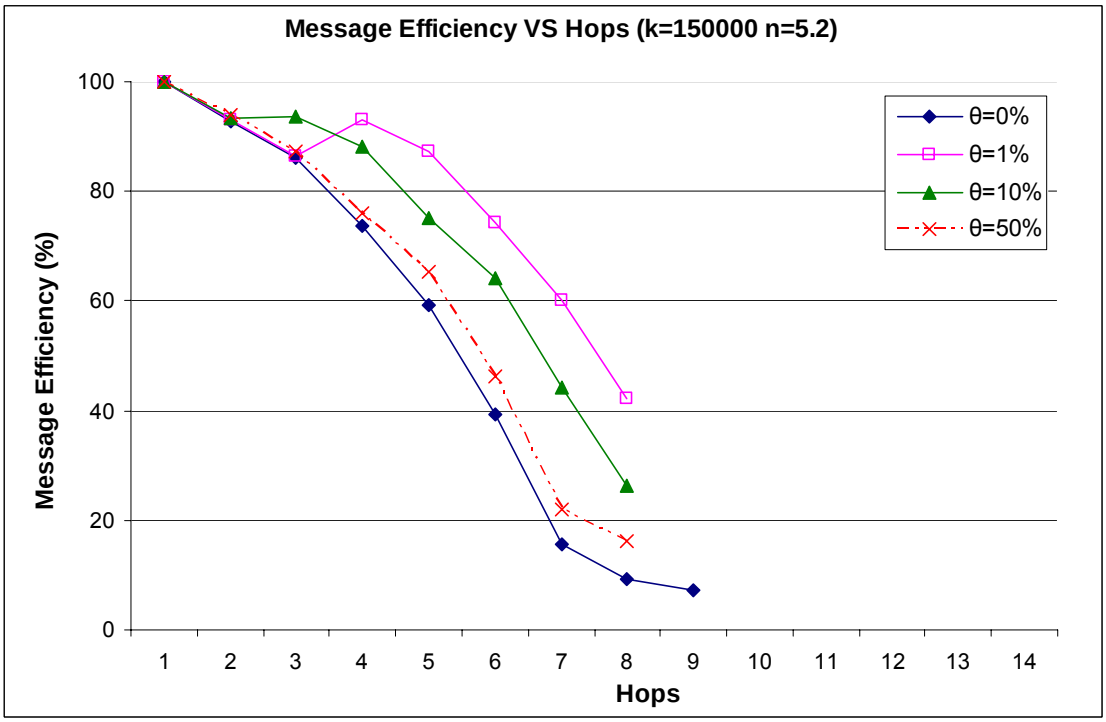


Figure 27. Message Efficiency in topology 2 with k=150000 at various θ values

4.4.4.3 Discussion

The experimental results have quantified the performance of the proposed heuristic to improve the scalability of Orion in more detail. In general, the existence of SeCOM of variable sizes can reduce the hop count to reach the destination DG. The reduction is more drastic when the SeCOM size is smaller. However, it is clear that the maintenance overhead of a smaller-sized SeCOM can be high. So, there is a need to strike a balance between the hop count to reach the destination DG and the SeCOM maintenance overhead.

The motivation for introducing SeCOM in Orion is to reduce message redundancy. The series of analysis clearly showed, with SeCOM, only a small number of DGs needs to participate in the message forwarding process, as compared to more than 80% of participating DGs in Orion without SeCOM. SeCOM, in particular SeCOM of size $\theta < 10\%$, can also maintain high message efficiency. High message efficiency means that in each forwarding hop, the messages are more likely to reach a DG that has not been visited before. Therefore, it is less likely for a DG to process the duplicated messages multiple times if message efficiency remains high for most part of the forwarding process.

The series of analysis also shows that SeCOM size should be kept small in order to maximize the benefit of deploying SeCOM. Although the size of SeCOM depends on the availability of context information and cannot be easily controlled otherwise, we may carefully design the membership classification function (i.e. the context classification function and query classification function) in order that popular context information can be classified at higher granularity. However, the overhead incurred due to maintaining small-size SeCOM should also be considered in the process.

Finally, the analysis results for both topologies show similar trend in most experimental cases. This is encouraging because it shows the robustness of the P2P-based architecture, which is able to give similar performance even if the underlying topology changes. When comparing the two topologies, it is obvious that topology 2 (with average outdegree $n = 5.2$) enjoys up to 45% less hop count in trying to reach the destination DG, at the expense of 5%-12% more DGs participating in the relay of query messages, as well as decreasing message efficiency at each hop level

Some parameter considerations are omitted in this simulation, such as nodes coming into and leaving the network, latency for each link, processing load on each node, etc. Furthermore, the DGs in each smart space will be connected at different bandwidths from low to high during actual deployment. However, the absolute numbers that we observe in these simulation results can be used to reflect and compare the tradeoffs between different parameters and different approaches. It helps us to understand the fundamental properties of the various techniques we apply in Orion.

4.5 Chapter Summary

The distributed DGs in Orion can self-organize into an unstructured-based P2P message routing overlay network. Various techniques, such as Iterative Deepening Search (IDS) and Semantic Community (SeCOM), are incorporated for the main purpose of reducing message redundancy when performing context discovery operations using the flooding-based search mechanism. Algorithms were derived to carry out the various network operations in Orion, such as the forwarding of messages through IDS, joining and leaving of SeCOM, and search operations for supporting the context lookup events. Through simulations, the various performance metrics of the Orion P2P network are evaluated.

CHAPTER 5 MATCHMAKING IN ORION

In context discovery process, a context provider announces its existence by putting up a *context advertisement*. On the other hand, a context requester specifies the discovery requirements in a *context lookup query* during the searching of the appropriate context providers. A match between the advertisement and the query will result in the *matchmaking* between the context requester and the relevant context provider. In this chapter, we look into the ontological representation model for context advertisement, as well as the semantic matching technique derived for the matchmaking process.

5.1 What is Matchmaking?

Matchmaking is the process in which a party is put in contract with potential counterparts [65]. The matchmaking process acts on the match of interest that is required by one party and provided by another. Matchmaking in Orion introduces a context requester to the appropriate context provider during the context lookup operations.

Figure 28 gives an overview of the matchmaking process. In a context provider, the set of context information $\boldsymbol{I_x}$, that it can provide is stored and maintained in a knowledge base known as the *context knowledge base*. The existence of the provider and the $\boldsymbol{I_x}$ it provides can be made publicly known through the announcement of the *context advertisement* $\boldsymbol{ad_x}$. $\boldsymbol{ad_x}$ is a meta-context abstraction and other relevant attributes of the context information set a context provider can provide. On the other hand, a context requester prepares two types of queries. The first one is the *context retrieval query* $\boldsymbol{q_y}$ for querying the information set in the context knowledgebBase of the relevant context provider. The second query type is the context lookup query $\boldsymbol{l_y}$ that delineates the

matching criteria that can be extracted from q_y, such as meta-context categories and context quality requirements, which must be matched in the process of discovering the appropriate context provider.

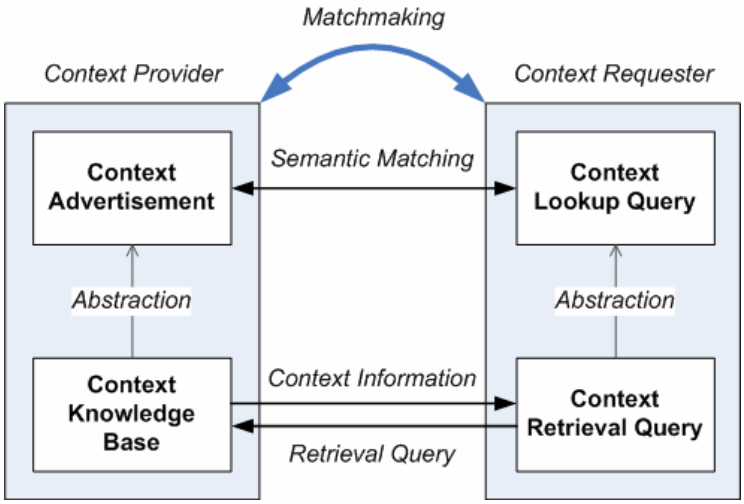


Figure 28. Matchmaking between context requester and context provider

In Orion, the *context discovery function*, $f_d : l_y \times AD \rightarrow id_x$, where $AD \subseteq \bigcup_{x=1}^{m} ad_x$, handles the matching of l_y against a set of registered ad_x. Each and every Discovery Gateway (DG) executes f_d based on its own set of registered advertisement localized to its residing smart space. The outcome is the identity, id_x, of a suitable context provider, based on its semantic similarities to the lookup requirements. To properly execute f_d, we first need to properly represent the context information. Then, based on the representation model, matching techniques can be applied. These two elements, *context representation* and *matching techniques*, constitute the elements of a successful matchmaking process.

5.1.1 Element 1 – Context Representation

In context-aware computing, various context models are proposed in order to represent the wide spectrum of context information in smart spaces and to facilitate context

sharing and interoperability among heterogeneous context resources. Strang and Linnhoff-Popien classified context modeling approaches into the following 6 categories: *Key-Value Model*, *Markup Scheme Model*, *Graphical Model*, *Object Oriented Model*, *Logic Based Model*, and *Ontology Based Model* [73].

In Orion, ontology based modeling technique is adopted for its well-known capabilities in expressive representation, logic inference, as well as knowledge sharing and reuse. The term ontology in Computer Science refers to the formal, explicit description of concepts, which are often conceived as a set of classes, relations, instances, functions and axioms [55]. The use of ontology enables the heterogeneous context resources to communicate and exchange information. Shared ontology defines a common understanding of specific terms, and this make it possible to communicate between systems on a semantic level.

The Context Advertisement Ontology (CoAO) is developed for modeling context advertisement. We use the W3C's recommendation for Semantic Web ontology language, Web Ontology Language (OWL) [56], for building CoAO. OWL offers expressive vocabulary for annotating the semantics of a contextual entity and its properties, as well as relationships with other entities. For example, properties of a class is described using either `owl:DatatypeProperty` for specifying string lateral and numeric value, or `owl:ObjectProperty` for relating to other ontology instances. OWL also has a semantic equivalence to description logics, which allows for consistency checking and contextual reasoning using inference engine developed for description logics.

5.1.2 Element 2 – Matching Techniques

With context advertisement properly described using ontological representation vocabularies, matching between lookup query and advertisement set is carried out using ontology-based semantic matching techniques. Semantic matching matches the concepts represented in the ontology, rather than mere string matching based on similarity in syntactic label. Semantic matching is important in ad-hoc environments, such as smart spaces, for interoperating the spontaneous and heterogeneous resources without a prior knowledge about each other.

Trastour *et al*. define the semantic matching algorithm for determining whether there is a match between two concepts *C1* and *C2* while performing matchmaking in ontology-based service description [65]. The algorithm concludes that "*C1* matches *C2*" if:

- *C1* is equivalent to *C2*, or
- *C1* is a sub-concept of *C2*, or
- *C1* is a super-concept of a concept subsumed by *C2*, or
- *C1* is a sub-concept of a direct super-concept of *C2* whose intersection with *C2* is satisfiable

Ranganathan *et al*. enhance the algorithm by associating certain similarity-level in ascending order for the 4 matching criteria, such that the search result can be efficiently filtered based on different similarity-level requirements [74].

By adapting the concept matching algorithm, together with the concept ontology defined in the CoAO, we propose a set of matching rules that decides what constitutes

a match between a lookup query and the "relevant" advertisement description. The outcome is a successful discovery of an appropriate context provider.

5.2 Representation Model

5.2.1 Context Advertisement

The vocabularies that compose a context advertisement are defined in the ontological advertisement template, known as the *Context Advertisement Ontology* (CoAO). Each advertisement announced by a context provider is an instance of the CoAO that outlines the profile of the provider, types of context information generated, ways of retrieving the information and choices of quality preferences for accurate matching. The 5 main classes in CoAO that model such advertisement attributes include *ContextProvider*, *ProviderProfile*, *ContextDomain*, *ContextQuality* and *AccessModel*. Figure 29 presents a graph representation that shows a snippet of the CoAO. The complete CoAO definition in OWL representation is available in Appendix A.

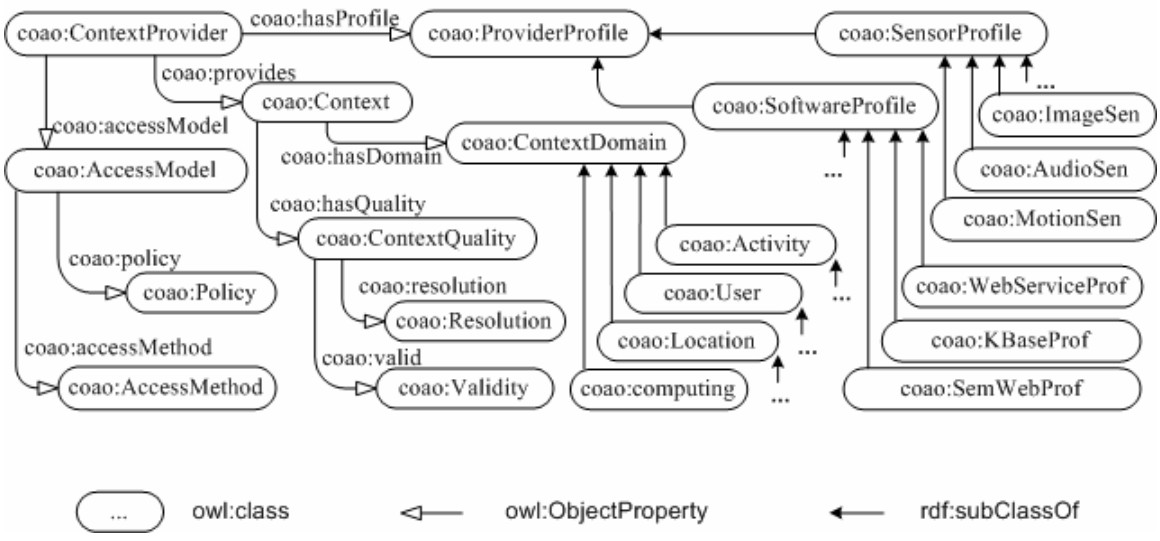


Figure 29. Graph representation showing fragment of Context Advertisement Ontology (CoAO)

A brief description of the 5 main classes is given next:

- **ContextProvider** – This class conceptualizes a context provider instance. The associated class properties include various identification attributes, as well as the locality of the residing space. Context provider identification is a uniquely assigned URN[17] using the Orion namespace.

- **ProviderProfile** – This class abstracts several possible types of context provider categories available in smart spaces. Subclasses *SensorProfile* and *SoftwareProfile* represent hardware-based provider and software-based provider respectively. Each subclass is associated with profile characteristic properties. For example, characteristic properties of the *SensorProfile* include physical location, sensor type, firmware version, etc.

- **ContextDomain** - *ContextDomain* class outlines the type and classification of the provided context information. The *Upper Level Context Ontology* (ULCO) defined in the Context Ontology model proposed in [9] is adopted in defining the hierarchical class relations in the *ContextDomain* class. ULCO is the common ontology for context information across different smart spaces, specifying three classes of real-world context (user, location, and computing entity) and one class of conceptual context (activity) [9]. ULCO can be suitably extended with class and property inheritance to customize the context requirements of every different smart spaces.

- **ContextQuality** – *ContextQuality* captures nonfunctional attributes that explains the context information quality preferences, such as correctness probability, provider capacity and communication cost. It links to the

[17] Refer to RFC2141: URN Syntax

Resolution class to denote the granularity of information, such as the measurement unit and value precision, and also the *Validity* class to answer the availability duration and updating frequency of the information. Context requesters express quality preferences in their lookup query to assist in the production of accurate matchmaking.

- **AccessModel** – This class describes the protocols used for establishing communication upon successful lookup. The *AccessMethod* subclass specifies communication protocol (e.g. HTTP, SOAP, etc) and port number to communicate with the context provider. The *AccessModel* also relates to the *Policy* class, which defines criteria for access rights based on privacy and trust considerations.

The CoAO presents an advertisement template that allows a context provider to advertise various meta-contexts that best describe the context it provides. As an illustration, Figure 30 shows an example context advertisement published by a road traffic monitoring system that monitors the road activity context in the *Clementi* district (i.e. `<spaceLocation rdf:resource="&hlt;Clementi"/>`). From the published advertisement, it is clear that the monitoring system has a sensor-based profile with sampling rate of 60 seconds. The current set of context information is valid from 8:15am onwards for a duration of 600 seconds, and the correctness probability is 0.97. Finally, to retrieve the updated context information, the monitoring system can be accessed via HTTP communication channel with the URI "`http://road.ex.org:19800`".

```
<rdf:RDF
   …
  xmlns:htl="http://.../HierarchicalLocationTaxonomy#"
  xmlns:ulco="http://.../SemanticSpace/ulco#"
  xmlns="http://.../Orion/coao#" >
   …
  <ContextProvider rdf:ID="ClementiTrafficMonitor">
     <providerID> urn:orion:xxxxxxxxxxxx </providerID>
     <spaceLocation rdf:resource="&hlt;Clementi "/>
     <hasProfile rdf:resource="#SP" />
     <provides rdf:resource="#context"/>
     <accessModel rdf:resource="#aModel"/>
  </ContextProvider>

  <SensorProfile rdf:ID="SP">
     <samplingRate> 10.0 </samplingRate>
     <samplingUnit rdf:resource="#Second" />
        …
  </SensorProfile>

  <Context rdf:ID="context">
     <hasDomain rdf:resource="#roadActivity"/>
     <hasQuality rdf:resource="#quality"/>
  </Context>

  <RoadActivity rdf:ID="roadActivity"/>

  <ContextQuality rdf:ID="quality">
      <correctness> 0.97 </correctness>
         …
      <valid  rdf:resource="#validity"/>
  </ContextQuality>

  <Validity rdf:ID="validity">
      <validFrom>2005-06-09T08:15:59</validFrom>
      <validPeriod>600</validPeriod>
  </Validity>

  <AccessModel rdf:ID="aModel">
      <accessMethod rdf:resource="#aMethod"/>
  </AccessModel>

  <HTTPAccess rdf:ID="aMethod">
      <Protocol>HTTP</Protocol>
      <HTTPURL>http://road.ex.org </HTTPURL>
      <PortNumber> 19800 </PortNumber>
  </AccessMethod>
   …

</rdf:RDF>
```

Figure 30. Context advertisement (XML representation) published by a road traffic monitoring system in Clementi district

5.2.2 Context Lookup Query

The context advertisement OWL description is maintained in the *Advertisement Cache* of the local space DG as a set of RDF context triples, i.e. (`subject`, `predicate`,

`object)` triple. Each triple outlines the relational property (i.e. `predicate`) of a resource (i.e. `subject`) with an `object` that can be either another resource or a certain value. Using the context advertisement in Figure 30 as an example, the triple set in the Advertisement Cache that the advertisement is registered with would contain the context triples shown below:

```
...
('ClementiTrafficMonitor', rdf:type, coao:ContextProvider),
('ClementiTrafficMonitor', coao:providerID, urn:orion:xxxxxxxx),
('ClementiTrafficMonitor', coao:spaceLocation, hlt:Clementi),
('ClementiTrafficMonitor', coao:hasProfile, 'SP'),
('SP', rdf:type, coao:SensorProfile),
('SP', coao:samplingRate, 10)
('SP', coao:samplingUnit, coao:Second)
...
```

As a result, the context lookup query can make use of the triple pattern query specification in a RDQL query language. RDQL (RDF Data Query Language) [69] is the *de facto* reference implementation of RDF query language. It is a SQL-styled query statement used to extract triple information from a RDF graph (i.e RDF triple set) based on a list of triple patterns. Each triple pattern consists of named variables and RDF values (i.e. URIs or literals). For example, matching of triple pattern "`... (?x, rdf:type, ns:User),(?x, ns:hasName, 'Bob')...`" with the triple set in Advertisement Cache will result in the variable `?x` to get the return value of all matching entity of type `ns:User` that has the attribute name "`Bob`". Additional set of constraints can be specified to limit the value range and type of the variables. The detailed grammar of RDQL is listed in Appendix B.

Context lookup query is built upon the RDQL query language to allow context requester to specify the criteria for a match in terms of triple pattern with one or more named variables. Figure 31 shows a self-explanatory context lookup query that is used to search for a road traffic monitoring system that observes the traffic condition in

Clementi (refer to example in Figure 30 for the relevant context advertisement). The query outcomes `?id`, `?x` and `?y` are respectively the *context provider identification*, *access URL* and *port* of the matching context provider respectively. Reasoning technique is applied to deduce implicit relations before the pattern matching begins. For example `<coao:validUntil>` can be deduced from `<coao:validFrom>` and `<coao:validPeriod>`. The rules for matchmaking decision will be further elaborated in the next section.

```
SELECT ?id ?x ?y
     WHERE
      (?p, <rdf:type>, <coao:ContextProvider>),
      (?p, <coao:providerID>, ?id),
      (?p, <coao:spaceLocation>, <hlt:Clementi>),
      (?p, <coao:hasProfile>, ?profile),
      (?profile, <rdf:type>,  <coao:SensorProfile>),
      (?p, <coao:provides>, ?context),
      (?context, <coao:hasDomain >, ?domain),
      (?domain, <rdf:type>, <coao:RoadActivity>),
      (?context, <coao:hasQuality>, ?quality),
      (?quality, <coao:validUntil>, ?t),
      (?p, <coao:hasAccessURL>, ?x),
      (?p, <coao:hasAccessPort>, ?y)
 AND
      (t > 2004-06-09T08:20:00)
      && (t < 2004-06-09T08:2 5:00)
 USING
      coao FOR <http://.../Orion/coao#>
      hlt  FOR <http://.../HierarchicalLocationTaxonomy#>
      rdf  FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#">
```

Figure 31. Context lookup query for discovering context provider that provides road traffic condition context in Clementi

5.3 Semantic Matching

Semantic matching ensures the successful matching of concepts between the context lookup query and the relevant context advertisement. Such process fits into the notion of *semantic discovery* for pervasive computing entities as described in [75] and [74], where the semantic discovery process is classified into three phases:

1. Discovery of all suitable classes that match a query
2. Discovery of all the entity instances of these classes
3. Filtering the instances to match the exact query

The 3-phase semantic discovery process resembles the matching procedure when querying over the semantics similarities of the ontological description. We therefore adapt the algorithm into a two-step semantic matching procedure in Orion. Step 1 encompasses the phase 1 and phase 2 of the semantic discovery. It identifies and selects the relevant subset of context advertisement triples in the Advertisement Cache which matches the context domain requirement specified in the triple pattern. And, in Step 2, the most appropriate context provider, among those context providers found in the previous step, is chosen based on various selection heuristic.

To illustrate the two-step semantic matching procedure, we consider an Advertisement Cache (AC) registered with ***X*** context advertisements. By assuming a context provider to publish only a single context advertisement to local space DG, the AC would have ***X*** instances of *ContextProvider* class. We therefore expect ***X*** subset of triple groups available in the AC, with each triple group containing an instance of a *ContextProvider* class and its associated set of context advertisement triples (see Figure 32).

We describe the two-step semantic matching procedure in the following sections.

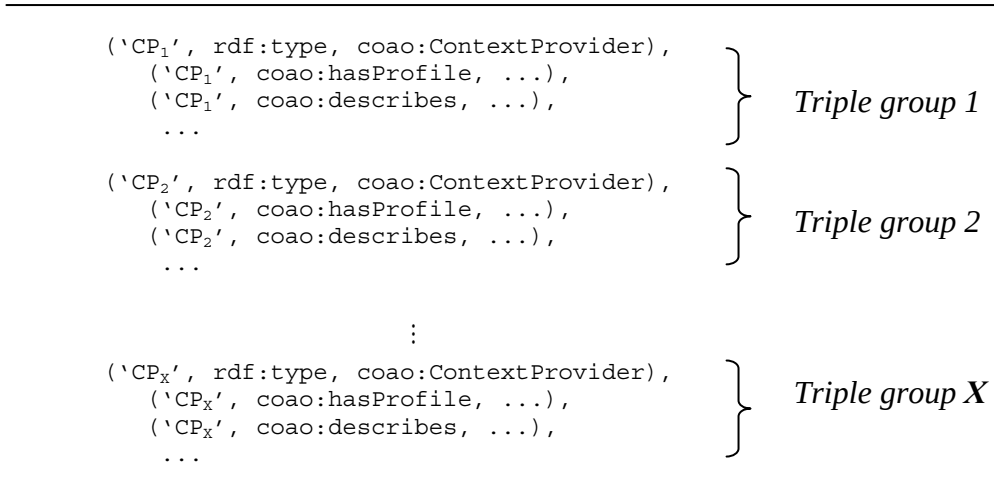


Figure 32. An Advertisement Cache (AC) containing ***X*** subset of triple groups

5.3.1 Step-1: Identifying the Triple Groups Having Domain Class Equivalence

Step-1 in semantic matching is interested in locating the relevant triple groups that match the context domain requirement. This is achieved by semantically matching the context domain of interest specified in the lookup query with the context domain registered in the advertisement. The context domain of interest is made clear in the triple pattern (`?x, coao:provides,` $context_a$), where $context_a$ is the context domain of the context information that the context requester is searching for. Similarly, the *ContextDomain* class in the CoAO allows the context providers to specify their related context domain in the context advertisement. The outcome of Step-1 is such that the identified one or more triple groups contains the registered context domain that poses either *exact class equivalence* or *subsumption class equivalence* with the context domain of interest specified by the context requester.

Exact class equivalence denotes two classes are exactly identical, such that both classes are mapped to the same class definition. For example, the class `ulco:User` in the ULCO ontology [9] and the class `soupa:Person` in the SOUPA ontology [57]

have exact class equality because they are referring to the same human entity in the smart space.

On the other hand, *subsumption class equivalence* implies that two domain classes posses subsumption relations to one another. Specifically, for two domain classes, *subsumption class equivalence* happens when one domain class is the superclass of another. This is an essential consideration when performing semantic matching. For example, if the context requester is searching for context provider that can reveal location of devices (e.g. position of devices) in the house, the lookup query would specify the domain of interest to be "*ObjectIndoorLocationContext*". As a result, the context providers that are able to present object tracking context information within the "*IndoorLocationContext*" domain (where "*ObjectIndoorLocationContext*" is a subclass of "*IndoorLocationContext*") can be a possible candidate context provider in the discovery process.

However, to preserve the hierarchical granularity of the domain ontology, *subsumption class equivalence* does not apply to context domain which belongs to the *Upper-Level Domain* (comprises the top level "ContextDomain" and followed by the second-level major domains including "LocationContext", "UserContext", "ActivityContext" and "ComputingEntityContext"). As a result, both context provider and context requester should not annotate domain information based on the context domains that belong to the Upper-Level Domain group.

Figure 33 presents the various possible scenarios of exact and subsumption class equivalence in the ContextDomain hierarchical ontology. Basically, two domain classes are of exact class equivalence relations when both of them have exactly the same path (i.e. total overlapping) that leads to an Upper-level domain class. Whereas,

when one domain class is placed in the middle of the path originated from the other domain class that leads to an Upper-level domain class, the two domain classes are of subsumption class equivalence. Finally, if the two paths started from two domain classes contain portions of the path segment which are not overlapped, we know that the two domains are non class equivalence. This is reflected in Algorithm 10 that is derived to identify class equivalence between two context domain classes.

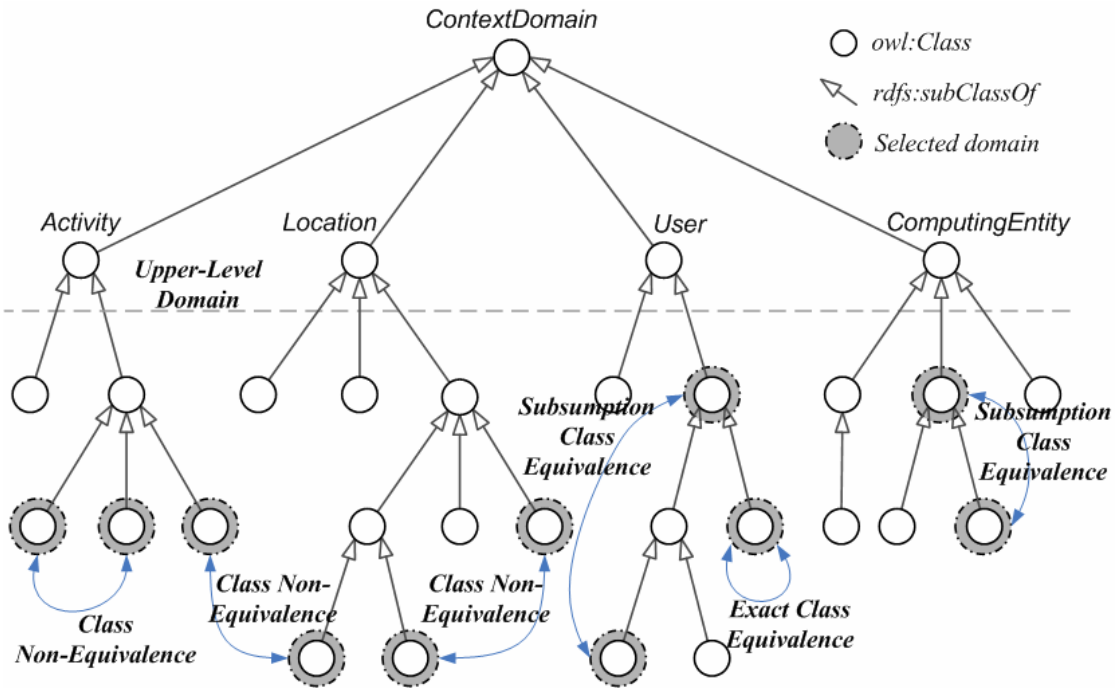


Figure 33. Various scenarios of class equivalence and non-equivalence between classes in the context domain hierarchical ontology

Algorithm 10. Matching rules for identifying class equivalence

Input: triple pattern (?x, coao:describes, $context_a$),
        triple (cp_i, coao:describes, $context_i$)
Output: ***True*** if $context_i$ and $context_a$ are class equivalence.
Procedure: Equivalence_Match
Begin
c_1 ← (?x, coao:describes, $context_a$)
c_2 ← (cp_i, coao:describes, $context_i$)
If ExactEquality (c_1, c_2) **then**
    Return ***True***
Else
    upper_level_domain_classes ← Get_Domain_Class (coao:ContextDomain)
    c1_superclass_list ← SuperclassOf (c_1) \ *upper_level_domain_classes*
    For each $c_1^s \in$ *c1_superclass_list*
        If ExactEquality (c_1^s, c_2) **then**
            Return **True**
    c2_superclass_list ← SuperclassOf (c_2) \ *upper_level_domain_classes*
    For each $c_2^s \in$ *c2_superclass_list*
        If ExactEquality (c_2^s, c_1) **then**
            Return ***True***
Return ***False***
End.

5.3.2 Step-2: Selecting the Most Appropriate Context Provider

If a relevant context provider exists in the smart space, at least one triple group will be identified in Step-1. Nonetheless, no triple groups being identified simply means none of the context providers in the smart space is qualified for providing the requested context information, and therefore the lookup query can be forwarded to the neighbours DGs directly.

The identified triple groups in Step-1 will need to be examined in Step-2 in order to ensure the appropriateness of the context provider to provide what is asked for. Triple matching technique is applied in matching the triple pattern in the lookup query against the context triples in each identified triple group. The various properties stated in the *ContextQuality* class and *ProviderProfile* class need to be matched in the triple matching process. The condition statements in the RDQL can also filter the irrelevant

matching results. We adopt a conservative approach whereby all triple patterns must be able to find the exact matching in a triple group before the context provider that registers the triple group advertisement is reckoned as the appropriate context provider. Consequently, the more triple patterns that a context lookup query specifies, the more accurate the search result would be, but it also causes probability of a successful lookup to be lower.

The exemplary context advertisement and context lookup query presented in Figure 30 and Figure 31 respectively can provide a walkthrough of the semantic matching process. In Step-1, the triple group that contains the context triples for the published advertisement is chosen as a result of the exact class equivalence in the "RoadActivity" context domain. Subsequently, in Step-2, all the named variables specified in the triple pattern can be successfully matched to the context triples in the selected triple group. Therefore, a successful matchmaking is established.

5.4 Chapter Summary

The key to a successful matchmaking lies in the ability to match the context lookup query with the most suitable context advertisement. In this chapter, we overcome the challenge by the proposal to use a semantic matching technique based on class equivalence relations. To facilitate the semantic matching process, we presented an ontology-based advertisement template for composing context advertisement using OWL ontology language. This matchmaking outcome ensured that the context domain of the discovered context information has class equivalence with the context domain of interest as specified by the context requester.

CHAPTER 6 IMPLEMENTATION

To validate the concepts of Orion in real life, a Discovery Gateway prototype is built and tested. In this chapter, we describe the implementation of the Orion architecture. The DG prototype is put into actions on the physical IP network infrastructure to measure the performance latency.

6.1 Implementation Methodology

In the simulation, we varied several parameters such as ONet size, SeCOM size and overlay network topology in order to evaluate the performance of the Orion architecture (see Chapter 4). To further verify the operations of Orion in real life, the Orion prototype is built.

A prototype of the Discovery Gateway (DG) is designed based on the DG architecture described in Section 3.3.2. The service-oriented architecture of the DG benefits from various open-source technologies. The *P2P Handler* is implemented with the *JXTA P2P framework*[18] to manage the neighbourhood information and to handle the message routing in the overlay network. In the *Discovery Service Handler*, we use the *Jena2 Semantic Web framework* [19] to process the OWL ontology, to maintain the Advertisement Cache knowledge base, as well as to perform reasoning over the ontology semantics.

The DG prototype is developed in Java programming environment. Therefore, the operation of the prototype is platform independent. It can operate on any computer that has a Java Virtual Machine (JVM), and a Network Interface Card. By running the DG

[18] Project JXTA, http://www.jxta.org
[19] Jena2 Semantic Web Framework, http://jena.sourceforge.net/

prototype on different desktops, we are able to implement the Orion P2P message routing overlay network using the public IP network infrastructure.

6.1.1 JXTA P2P Framework

JXTA (short for *juxtapose*) is a set of protocols that facilitates P2P communication over the existing physical network infrastructure [76]. It aids in the development and deployment of P2P applications and services without needing us to understand or manage the physical network topologies.

The smallest addressable entity in a JXTA network is a *peer* that implements one or more of the JXTA protocols. Peer resides in one or more *peer groups* (by default, all peers belong to the *Net Peer Group*) whose members have agreed upon a common set of services, such as enforcement of specific security policy, and sharing of certain specialized domain content. The peers are connected by *pipes*, which are the asynchronous and unidirectional communication channels for transferring messages (e.g. data strings, binary codes, documents, etc) to one another. A single pipe can have several endpoints that connect to the input pipe (the receiving end) and the output pipe (the sending end). To the peers, the pipe endpoints correspond to P2P network interfaces that can be used to send and receive message.

JXTA framework has six protocol suites that handle various P2P network operations. The protocols include the *Peer Discovery protocol*, *Peer Information protocol*, *Peer Resolver protocol*, *Peer Binding protocol*, *Endpoint Routing protocol*, and *Rendezvous protocol*. However, in prototyping the Orion architecture, not all protocol suites are utilized. Some protocol modules are also modified in order to adhere to the Orion architecture specifications. The JXTA P2P framework is mainly employed to provide the following functionalities:

- Maintaining DG peers information, such as peerID, update time, pipe advertisements, connection status, neighbourhood information, etc,
- Discovering potential neighbour DGs during bootstrap operations,
- Handling the physical network connection by establishing pipe connections with the neighbour DGs,
- Relaying messages during the search forwarding operations.

6.1.2 Jena2 Semantic Web Framework

Jena2 is an open-source Java programming framework for building Semantic Web applications [77]. It provides a set of APIs for the manipulation, storage, interpretation and query of RDF and OWL documents. These are accomplished by using its ability to support expressive RDF query, to perform generic rule-based inference, and to offer scalable persistent storage.

Internally, Jena2 manages the OWL ontologies as the *RDF Graph* data structures, where every two vertices joined by an edge in the graph represent a RDF triple. A rich set of APIs are provided to manipulate the RDF Graph, therefore enabling application programmers to easily gain access to the triple structure. The RDF Graph is stored as a graph model in the memory, and the model can be reasoned, via built-in or plug-in rule-based inference engine, in order to check for data consistency and to deduce implicit class relations and instances.

Jena2 supports RDF Data Query Language (RDQL) [69] for programmers to extract information from the RDF graph. One or more graph patterns, in the form (`subject`, `predicate`, `object`), are presented to the query engine in RDQL, and all possible

valid bindings of the variables in the patterns over the statements in the graph are returned.

In the Orion prototype, we benefited from the Jena2 Semantic Web framework in the following area:

- Storing the context advertisements as set of context triples,
- Checking for model consistency based on the defined CoAO ontology and the HLT ontology,
- Performing ontology reasoning to deduce subsumption relations between the class instances when checking for class equivalence, and to reason out transitive relations when extracting the SeCOM membership requirements,
- Performing triple matching during the semantic matching process.

6.2 Discovery Gateway Prototype

We design and implement the prototype of the Discovery Gateway in an object-oriented fashion. Though independent object components handle different tasks, cooperatively they execute the set of operations to support context publishing and context lookup. The schematic overview of the Discovery Gateway prototype architecture is presented in Figure 34. The rectangular boxes represent the various object components while their interactions are indicated by the arrows.

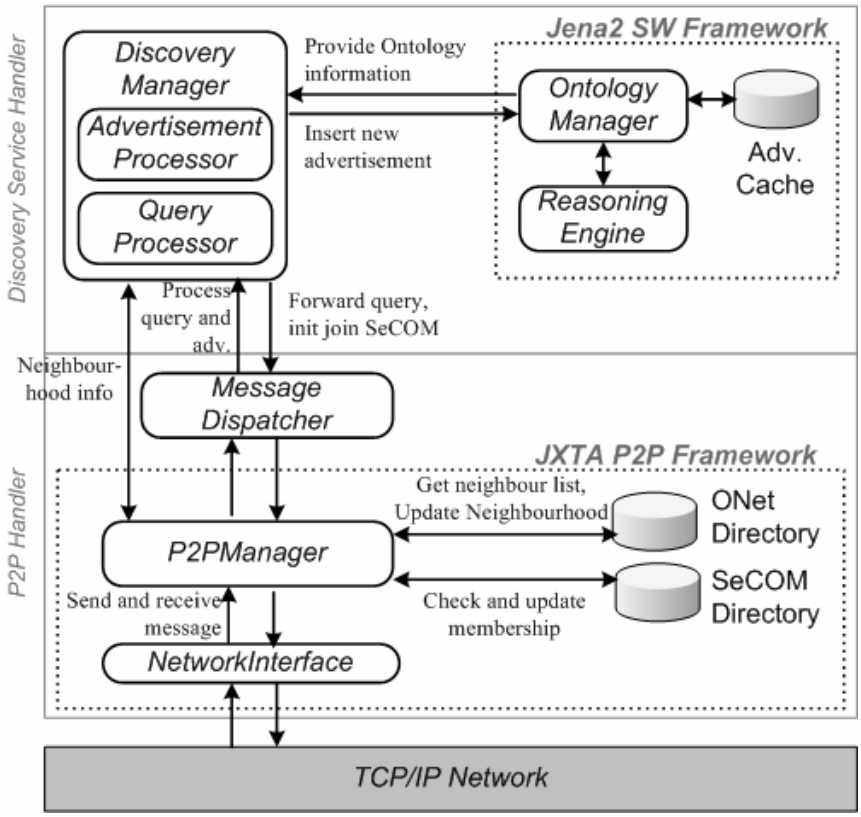


Figure 34. Discovery Gateway prototype architecture overview

The two-layer architecture presented in Section 3.3.2 is preserved in the prototype implementation. In the *P2P Handler* layer, the JXTA P2P framework is used for handling the P2P communication between the DG peers. The *NetworkInterface* component is responsible for message reception from and transmission to other DG peers over the public network infrastructure. Many features of the JXTA framework are relied upon. For example, the `net.jxta.discovery.DiscoveryListener` interface is implemented to respond to new DG node discovery event; `net.jxta.pipe.PipeMsgListener` interface is used to asynchronously handle message reception via the JXTA pipe established between the neighbour DGs; while `net.jxta.pipe.OutputPipe` interface is implemented to send messages to the neighbours via the Output pipes.

The *P2PManager* is the main component that takes care of the peer neighbourhood management. It implements the DG peer boostrap and leave process (in Section 4.1.1 and Section 4.1.2) as well as Algorithm 1 and 2 (in Chapter 4.1.3) for Iterative Deepening Search. It decides when and where the message is to be forwarded to (either to the upper level for query or advertisement processing, or to the neighbour peers in an IDS process).

For messages that need to be processed by the upper layer, the *MessageDispatcher* analyzes the message header to categorize the messages, and to dispatch them to the appropriate processing unit accordingly. Whenever a message has to be forwarded, the *MessageDispatcher* is also responsible for composing the appropriate message format before passing the message to the *P2PManager*.

In the *Discovery Service Handler* layer, two sets of object components are implemented. The first set handles the context discovery services, including the context advertisement and the context lookup. The *AdvertisementProcessor* executes the context publishing operations as outlined in Algorithm 7, while the *QueryProcessor* implements Algorithm 8 and 9 for the context lookup operations, as well as Algorithm 10 for the semantic matching procedure.

The second set consists of mainly objects that maintain and manipulate the ontology instances. It performs query and reasoning over the ontology knowledge base (e.g. the *Advertisement Cache*). The APIs provided by the Jena2 Semantic Web framework are used in this implementation. For example, the Triple Groups in the Advertisement Cache are stored and managed in the `com.hp.hpl.jena.rdf.model.Model` object; the RDQL query initialization and execution during the triple pattern matching in the matchmaking process is handled by the `com.hp.hpl.jena.rdql.QueryEngine` object.

To further illustrate the interactions between the object components in the prototype architecture, we analyze the operational sequences for handling the context publishing event. The interactions between the object components are clearly indicated in the sequence diagram in Figure 35. When the context provider publishes a context advertisement (step1), the published message will be received by the *NetworkInterface* and passed directly to the *P2PManager* (step2). The function `Analyze(msg<ad>)` decides whether the message has to be processed by the *Discovery Service Handler* layer (step3). Then the *MessageDispatcher* object is contacted (step4). Here, the message is assigned a unique messageID, which is active throughout the publishing event, and is cached locally to prepare for future use (step5). Subsequently, the message is inspected to determine the message type (step6). If it is then known that the message contains the context advertisement, the `ProcessAd(ad)` function in the *AdvertisementProcessor* object will be executed (step7). The advertisement registration procedures include updating the Advertisement Cache (step8-11), as well as extracting the membership information to determine the geo-location meta-context the context is having (step12-14). The request to perform *JoinSeCOM* operation is executed then (step15-19) (the details of *JoinSeCOM* operations that involve other DGs are not presented in the sequence diagram in Figure 35).

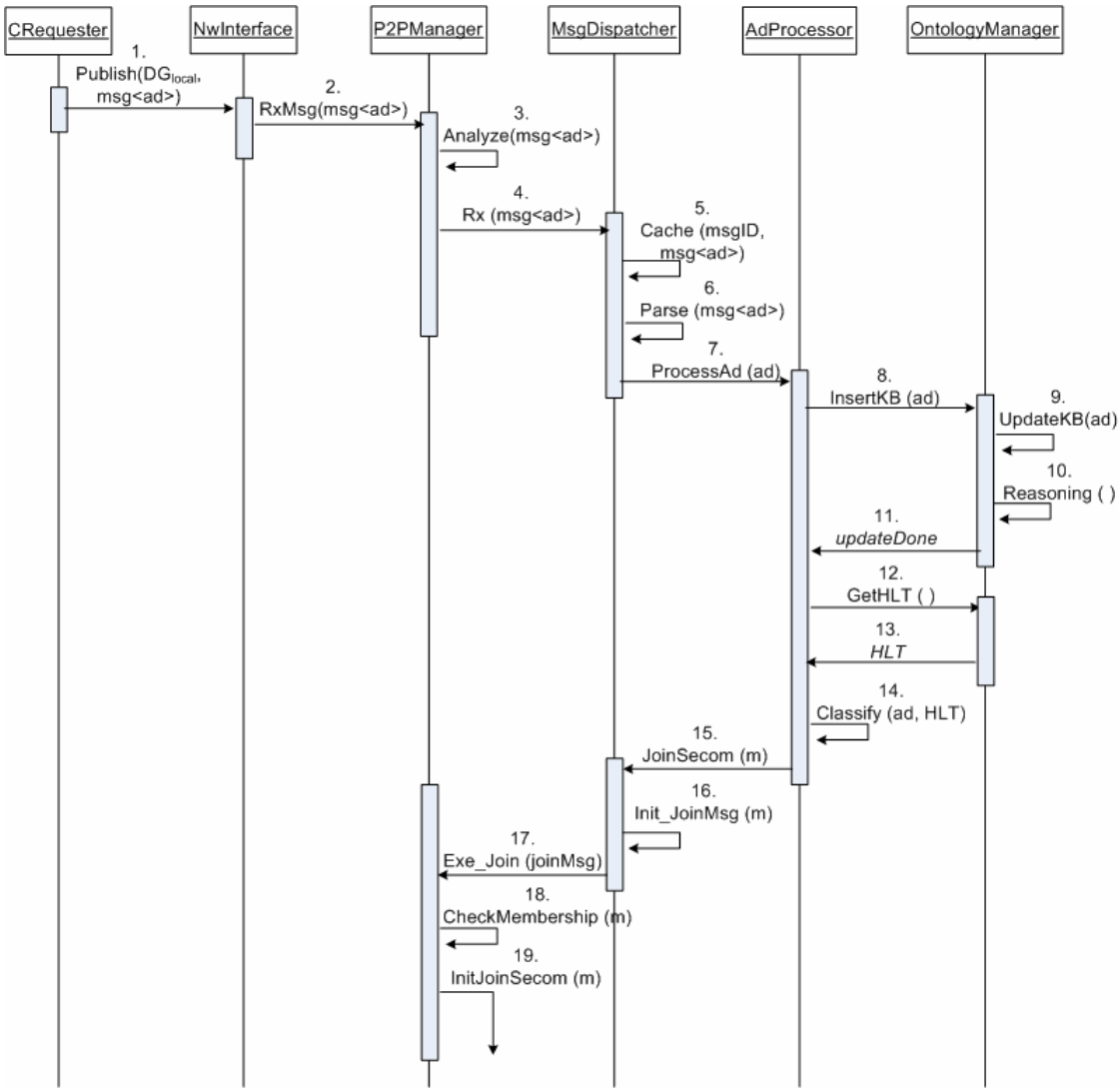


Figure 35. Sequence diagram shows the interactions between objects in handling context publishing event

The DG prototype is implemented on Java programming platform, and thus it is platform independent. It can operate on any computer that is equipped with a Java Virtual Machine (JVM), together with a Network Interface Card that can connect to the Internet.

6.3 Evaluation

The various performance metrics of Orion consisting of over thousands of DGs are evaluated in the simulation (refer to Section 4.4). In the prototype system, we continue the evaluation with emphasis on two areas: the query response time for a single DG, and the aggregated query response time for a DG placed several hops away. Query response time is defined in this evaluation as the time difference between the instant the local DG starts processing the lookup query and the instant the processing ends (i.e. when the access information of the discovered context provider is received).

6.3.1 Query Response Time Within Local Space

We are interested to find out the query response time for a local space DG to be able to resolve the lookup query without forwarding to its neighbours. We have developed a script that generates variable numbers of context triples (i.e. by generating different number of unique class and property instances) in the Advertisement Cache ontology knowledge base. We then evaluated the query response time at variable size of the knowledge base.

We carefully tweak the provider lookup requirement such that there is always one context advertisement in the Advertisement Cache that satisfies the lookup requirements. Upon a successful match, the IP address and the port number of the discovered context provider are returned and verified. We use the `getTime()` method in the Java j2sdk's *Date* class to obtain the system clock, and record the time spent on processing a query in the knowledge base. The results are compiled to produce the graph in Figure 36.

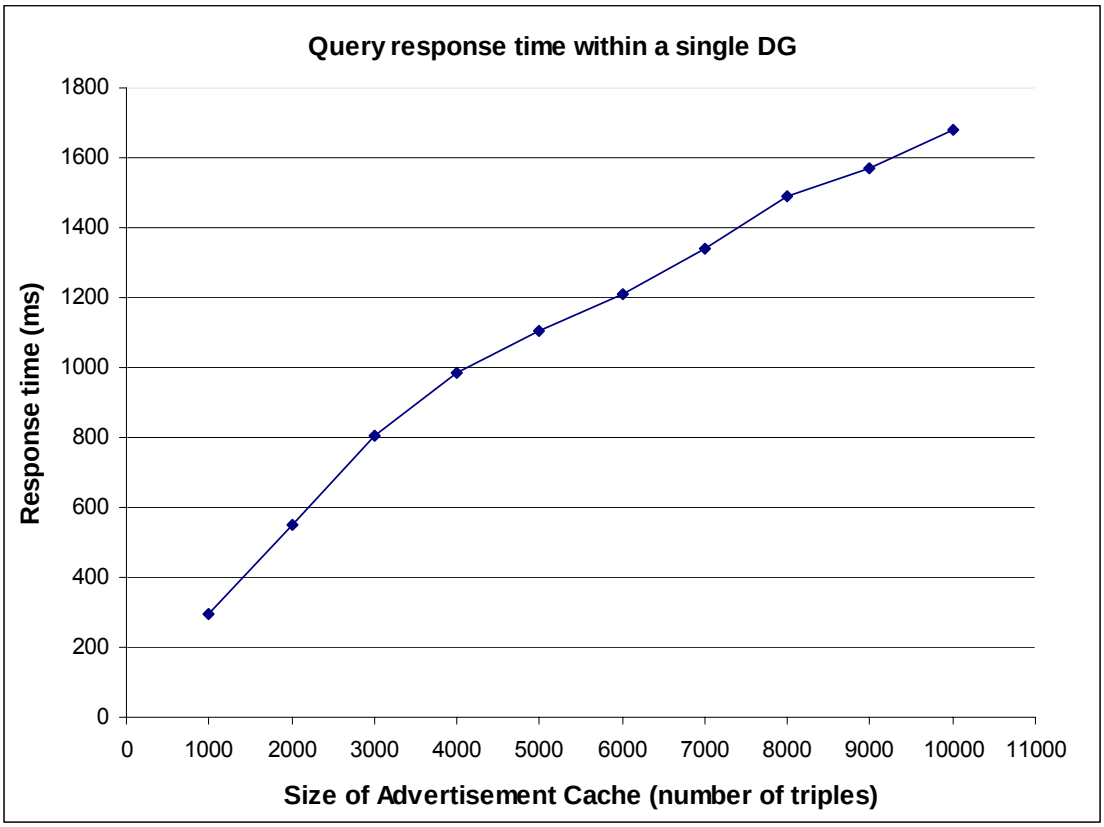


Figure 36. Query response time within a single smart space

The graph shows that the response time of DG is linear to the size of the ontology knowledge base. When the number of context triples increased from 1000 triples to 10000 triples, the response time steadily increases from 300ms to 1700ms.

In [9], we see that a Context Knowledge Base of a Semantic Space Server that manages all context information in a single workplace smart space can accumulate up to approximate 3000 context triples. Since context advertisement is an abstraction of the available context information in the smart space, we expect the triple counts of the Advertisement Cache knowledge base would be a lot smaller than the triple counts of the Context Knowledge Base. This also justifies the need to launch a context discovery operation to query the lightweight Advertisement Cache prior to querying the huge Context Knowledge Base. This is especially essential as the query is relayed from one

DG to another. Until there is a successful matchmaking of the context lookup query that takes place in the lightweight Advertisement Cache knowledge base, a DG will only need to process the context retrieval query in the Context Knowledge Base.

6.3.2 Query Response Time Across Multiple Spaces

In the second experiment, we would like to evaluate the query response time for a lookup query to be resolved by a DG placed in a smart space several hops away. The query response time therefore includes the lookup query processing time in each participating DG, and the aggregated propagation delay in all DG-to-DG overlay links.

We have chosen 9 desktops running on Windows XP operating systems to run the DG prototype. The desktops are placed at different locations in Singapore, and the particulars about the connection type and communication bandwidth are tabulated in Table 2. Each DG prototype maintains TCP/IP connection to 2 neighbour DGs, and an overlay network of 8 hops is formed on top of the IP networking public infrastructure (see Figure 37). The mixture of broadband network access services with different bandwidth as well as the dialup connection in DG node 5 resembles the bandwidth variation for the overlay links in a message routing overlay network, which is formed across the public network infrastructure.

Each of the 9 DGs maintains an Advertisement Cache of about 2000 context triples contributed by around 80 context advertisements. The average lookup query processing latency in each DG node is measured and tabulated in Table 3.

Table 2. Details of the DG prototype deployed for experiment 2

DG	IP Address	Physical Location	ISP *	Avg up-link b/w (kbps)	Avg down-link b/w (kbps)
1	59.189.27.65	Clementi Ave 5	SM	117	573
2	218.186.179.77	Clementi Ave 3	SM	109	452
3	218.186.66.101	Jurong West Ave 5	SM	97	398
4	218.186.74.140	Bukit Batok West Ave 5	SM	126	513
5	165.21.57.67	Stirling Road	D	36	45
6	202.156.186.85	Serangoon Ave1	SM	107	389
7	218.186.170.230	Queensway	SM	120	368
8	219.74.169.164	Amber Road	SB256	153	324
9	220.255.206.54	Hougang Ave 7	SB1500	168	416

* **SM**: Starhub MaxOnline 2000 **D**: dialup
SB1500: SingNet Broadband 1500 **SB256**: SingNet Broadband 256

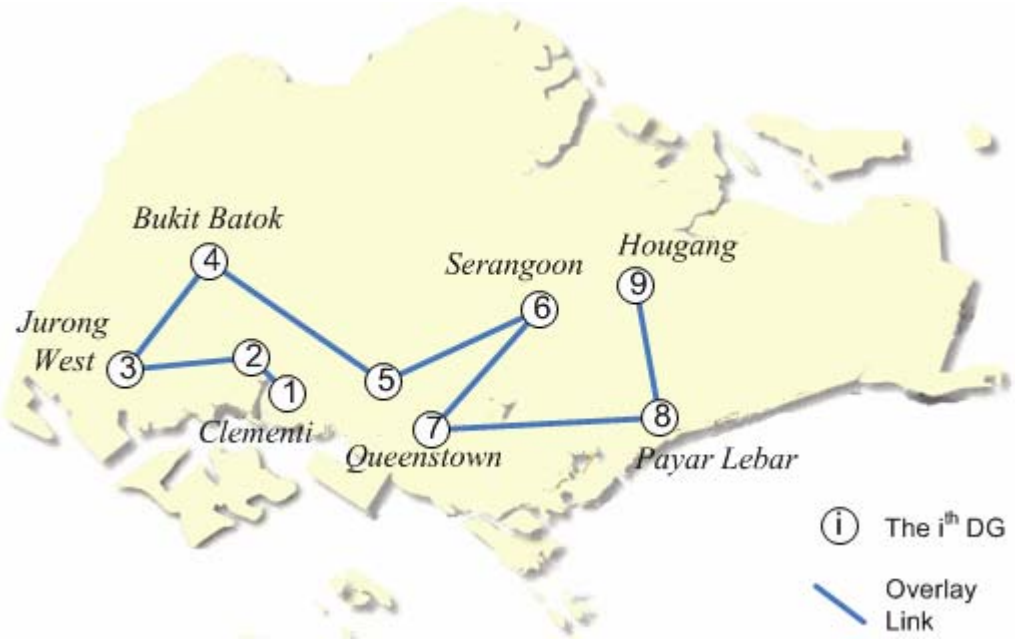


Figure 37. The topology created for evaluating query response time

Table 3. Average query processing latency in each DG prototype node

DG node *i*	Average query processing latency (ms)
1	343
2	403
3	611
4	412
5	373
6	294
7	365
8	421
9	639
Total	*3861*

DG node 1 in the overlay network serves as the sender node where a lookup query is initiated. On the other hand, DG node 9 stores a context advertisement that matches with the lookup query requirement. Therefore, starting from DG node 1, the query message is forwarded from node *i* to node (*i*+1). The Advertisement Cache lookup processing would have been unsuccessful except in node 9. When the query is finally resolved in DG node 9, a reply message is returned to DG 1 via a shortcut link established between them (the shortcut link is not shown in Figure 37). All DGs are assumed to be in the same SeCOM where the query can be resolved.

The overall query response time is recorded in DG node 1 and presented in the line chart in Figure 38. It can be observed that the response time fluctuated at around 5 seconds, where the respond time before 1200 hrs dropped slightly below 5 seconds while the rest were between 5 to 5.5 seconds. Since the size of the Advertisement Cache remains the same throughout the experiments, the fluctuation in the overall query response time is mainly contributed by the change of network link latency over the course of a day

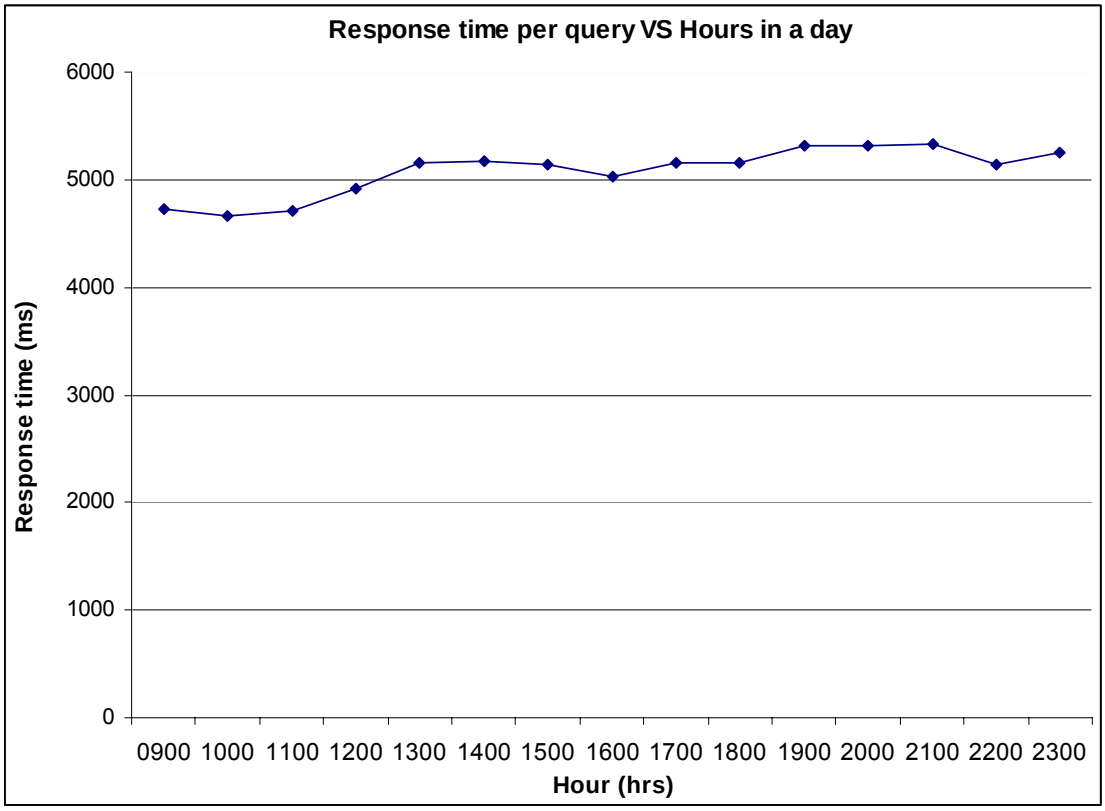


Figure 38. The query response time measured when query is resolved in a DG prototype that is 8 hops away from DG node 1.

As a result, we also measure the message transmission latency at each overlay link. At each time interval, DG node *i* performs a RTT probing to node *i*+1 in order to measure the round trip time (RTT) for *link*(*i*, *i*+1) (i.e. the overlay link connecting node *i* and node *i*+1). The shortcut link between node 9 and node 1 is denoted as *link*(9,1). By equally dividing the RTT of *link*(*i*, *i*+1) into half, we get a rough estimation of the one-way link latency between node *i* and node *i*+1.

The measured link latency is shown in the stacked histogram in Figure 39. The height of each coloured portion indicates the message transmission link latency, and the specific overlay links are differentiated by the colour scheme. The height of each stack is therefore the accumulated link latency for all the 9 overlay links set up in the experimental topology. It is observed that before 1200hrs, the overall network

transmission latency is well within the 800ms range. However, as the time progresses beyond 1200hrs, the overall link latency varies between 1100ms and 1400ms. The differences of about 75% in link latency between morning and evening is a direct result of different usage levels of the Internet network infrastructure at different periods of time.

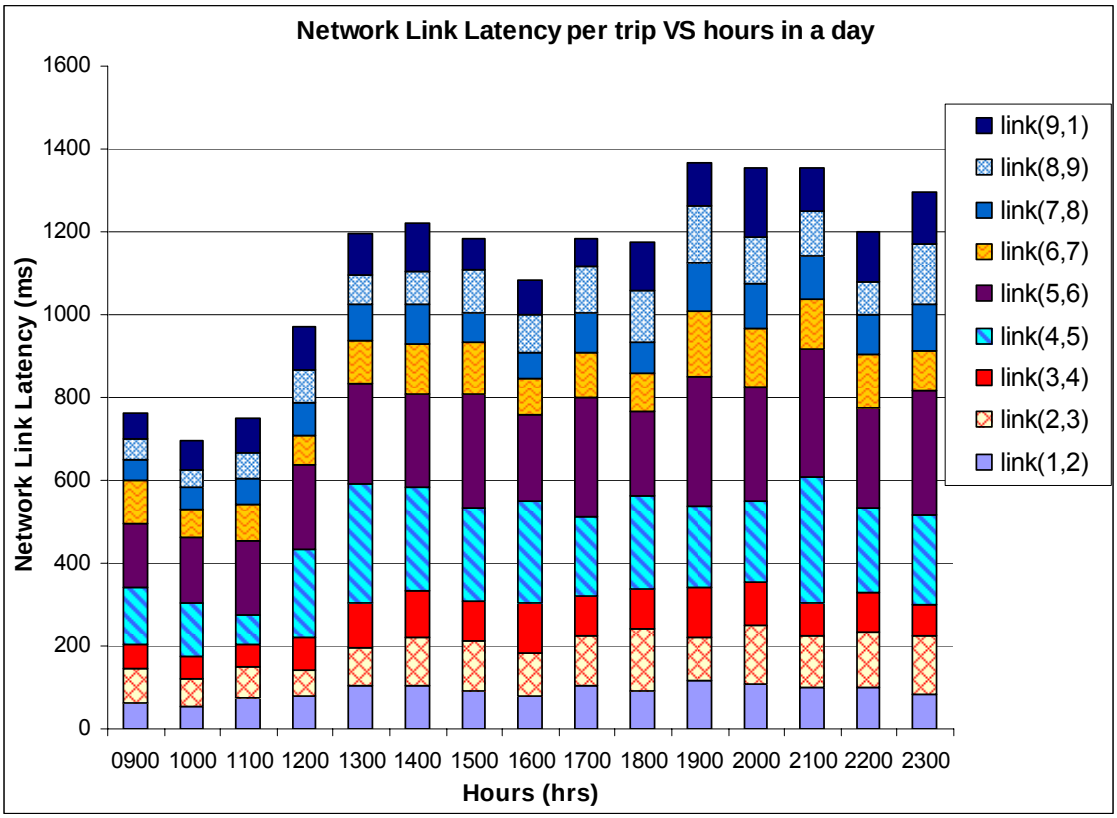


Figure 39. Message transmission link latency at each overlay link that contributes to the overall query response time

In this simulation, about 20%-30% of the overall query response time is contributed by the network link latency incurred in each overlay link. Therefore, although network link latency varies by about 75% throughout the day, the overall query response time only fluctuates at 18% difference (i.e. from 4.5ms to 5.5ms). This is the case when about 11% of the participating DGs (i.e. 1 out of 9) are connected with low-bandwidth

dial up connection. We expect the fluctuation to become larger when more low-bandwidth connections are involved.

6.4 Chapter Summary

In this chapter, we make use of the open-source JXTA P2P framework and Jena2 Semantic Web framework to build the prototype of Discovery Gateway. The prototype is put into action for measuring the query response time of a small scale overlay network laid across the Singapore island.

CHAPTER 7 CONCLUSION AND FUTURE WORK

7.1 Conclusion

Pervasive computing smart spaces are rich in context information produced by various interconnected sensors and software sources. To maximize usage of the context information widely distributed across different smart spaces, the context providers need to be efficiently tracked by the context-aware applications with minimal user intervention. This dissertation addressed the issue of system infrastructure needed to support the discovery of context information, in particular context discovery beyond local smart space boundaries.

A hybrid centralized-decentralized context discovery model was proposed. Compared to the traditional centralized model and broadcast-based model, the proposed model was superior in terms of its ability to handle high computational load and large information storage space requirement, to provide reliable discovery service, to ensure timely update of localized information, and to scale well beyond a single smart space boundary. The model was materialized in a P2P-based context discovery platform, named *Orion*. We focused on two areas in the Orion architecture, namely the searching in P2P network and the matchmaking procedure in each Discovery Gateway (DG).

The P2P message routing overlay network successfully formed a message communication platform to connect the context resource peers (i.e. both context providers and context requesters) to one another. To reduce duplicate message in the overlay network due to flooding-based search mechanism, Iterative Deepening Search (IDS) was introduced to limit the forwarding range, and Semantic Community

(SeCOM) was incorporated to confine the flooding of message within a subset of DGs that shared similar interest in the context they were registered with. A set of distributed algorithms were created as the heuristic in query forwarding. In our deployment, at least 60% of nodes (can go up to 95% if SeCOM size were small) were spared from taking part in the flooding (as compared to only 10% of spared nodes in pure flooding approach), and average hop counts to reach the destination DG were also lowered by at least 16% when the SeCOM size was kept very small (at 1% of the total ONet size). Therefore, Orion has successfully optimized the unstructured-based P2P message routing overlay network by reducing message redundancy and minimizing the number of hops to reach destination DG.

The second focus was the matchmaking process. We proposed an ontology-based advertisement template, known as the Context Advertisement Ontology (CoAO), to model the context advertisement with ontological descriptions, of which the semantics could be interpreted by the computer. Such semantics reasoning capabilities led to the matchmaking procedure based on semantic matching of the class equivalence between the lookup query and the advertisement set. This is a mechanism far superior compared to string matching approach, because we ensured that the matchmaking was done based on the similarity in semantics, rather than the resemblance of keyword string. For example, the string "room" could be interpreted differently by various computing agents, and ontology provided the semantics necessary for understanding the concept of "room" as how human would interpret it.

Finally, the DG prototype had not only verified the design, it also contributed as a development platform that other researchers could make use of. This would facilitate

new research initiatives in wide-area context management, which are greatly beneficial to mobile and wireless communication technologies.

7.2 Future Work

There are other challenging issues in inter-space context discovery, which are not dealt with in the current Orion architecture.

In a context-aware computing system, the distribution of different query type is not uniform. This means that some types of context information can be highly demanded, while some types can be unpopular. This poses performance bottleneck issue at the DG registered with highly demanded context information types because the query frequency will be much higher than DGs registered with unpopular context information types. As a result, the computational load is not distributed equally in all the DGs, and that affects the overall query response efficiency in Orion. To overcome this problem, one approach is to suitably replicate the Advertisement Cache of a DG and store each duplicate copy on one or more of the SeCOM member DG. Consequently, for *m* replications made, the query load at each DG will be reduced to 1/*m* of the initial load [78]. Different replication strategies in P2P network were studied in [30], [31], [78] and [79].

Methods to safeguard information privacy are also an issue that tops the to-do list. Context information with sensitive contents needs to be kept confidential, and only parties with sufficient clearance have rights to discover and retrieve them. This would prevent fraudulent use of context information by unauthorized users and misbehaving applications. We may extend the *Policy* class in the CoAO ontology to support *semantic policy language* used in the Semantic Web to define security requirements in

terms of *permissions*, *prohibitions*, *obligations* and *dispensation* [80]. Then, only context requesters with the proper rights defined in the context advertisements are allowed to gain access to the requested context information.

Other future work in the agenda includes expanding the CoAO to support more comprehensive classification of the provider's profile and context domain, imposing structured-based topology in SeCOM for efficient routing of query message, and incorporating leader election algorithms for automatic selection of new DG when the current DG fails to operate.

BIBLIOGRAPHY

[1] M. Weiser, "The Computer of the 21st Century", *Scientific American*, Vol. 265, No. 3, pp. 66-75, Janunary 1991.

[2] B. Brumitt, B. Meyers, J. Krumm, A. Kern, S. Shafer. "EasyLiving: Technologies for Intelligent Environments", In *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing*, Vol. 1927, pp. 12-29, Bristol, UK, September 25-27, 2000. Springer Verlag.

[3] M. Roman, *et al*., "Gaia: A Middleware Infrastructure to Enable Active Spaces", In *IEEE Pervasive Computing*, Vol. 1, No. 4, pp. 74-83, October-December 2002.

[4] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges", In *IEEE Personal Communications*, Vol. 8, No. 4, pp. 10-17, January – March, 2001.

[5] H. Lieberman, T. Selker, "Out of Context: Computer Systems that Adapts to, and Learn From, Context", *IBM Systems Journals*, Vol. 39, No. 3-4, pp. 617-632, 2000.

[6] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, W. V. Velde, "Advanced interaction in context". In Proceedings of *First International Symposium on Handheld and Ubiquitous Computing*, HUC'99, pp. 89-101, Karlsruhe, Germany, September 1999. Springer Verlag.

[7] A. K. Dey, "Understanding and Using Context", *Personal and Ubiquitous Computing Journal*, Vol. 5, No. 1, pp. 4-7, 2001.

[8] B. Schilit, N. Adams, R. Want, "Context-aware computing applications. In Proceedings of *IEEE Workshop on Mobile Computing Systems and Applications*, pp. 85-90, Santa Cruz, California, December 1994. IEEE Computer Society Press.

[9] X. Wang, J. S. Dong, C. Y. Chin, S. R. Hettiarachchi, D. Zhang, "Semantic Space: An Infrastructure for Smart Spaces", In *IEEE Pervasive Computing*, Vol. 3, pp. 3, pp. 32-39, July- September, 2004.

[10] J. Pascoe, "Adding generic contextual capabilities to wearable computers". In Proceedings of the *Second International Symposium on Wearable Computers*, Pittsburgh, Pennsylvania, October 1998. IEEE Computer Society Press.

[11] D. Siewiorek et. al., "SenSay: A Context-Aware Mobile Phone", *7th IEEE International Symposium on Wearable Computers*, October, 2003.

[12] M. Reinhard, Specht and I. Jaceniak, "Hippie: A Nomadic Information System", In *Proceedings of 1st International Symposium on Handheld and Ubiquitous Computing* (HUC '99), pp. 330-333, Karlsruhe, Gerrmany, 1999.

[13] K. Yang *et al*., "Network-centric context-aware service over integrated WLAN and GPRS Networks," In *14th IEE International Symposium on Personal, Indoor And Mobile Radio Communications*, Beijing, China, September 7-10, 2003.

[14] B. Schilit, M. Theimer, "Disseminating Active Map Information to Mobile Hosts", In *IEEE Network*, Vol. 8, No. 5, pp. 22-32, 1994.

[15] H. Yan, T. Selker, "Context-aware Office Assistant", In *Proceedings of the 2000 International Conference on Intelligent User Interfaces*, pp. 276-279, New Orleans, LA, January 2000. ACM Press.

[16] A. Schmidt, M. Beigl, H-W. Gellersen, "There is more to Context than Location", In *Computers & Graphics Journal*, Vol. 23, No.6, pp. 893-902, December 1999. Elsevier.

[17] B. L. Harrison, K.P. Fishkin, A. Gujar, C. Mochon, R. Want, "Squeeze Me, Hold Me, Tilt Me! An Exploration of Manipulative User Interfaces", In *Proceedings of the ACM SIGCHI conference on Human Factors in Computing Systems*, pp. 17-24 , Los Angeles, CA, USA, April 18-23, 1998.

[18] J. Rekimoto, "Tilting Operations for Small Screen Interfaces", In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, pp. 167-168, Seattle, Washington, USA, 1996.

[19] A. K. Dey, G. D. Abowd, D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications". anchor article in *Human-Computer Interaction (HCI) Journal*, Vol. 16, No. 2-4, pp. 97-166, 2001

[20] J. Hong, J. A. Landay, "An Infrastructure Approach to Context-Aware Computing", *Human-Computer Interaction* Journal, Vol. 16, No. 2-4, 2001.

[21] G. Chen, "Solar: Building a Context Fusion Network for Pervasive Computing", Ph.D. Dissertation, Department of Computer Science, Dartmouth College, August 2004.

[22] H. Chen, T. Finin, A. Joshi, “An Context Broker for Building Smart Meeting Rooms”, In *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium*, 2004 AAAI Spring Symposium. AAAI, March 2004.

[23] R. E. McGrath. “Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing”. Presentation to the Center for Excellence in Space Data and Information Science, UIUCDCS-R-2000-2154, April 2000

[24] C. Efstratiou, K. Cheverst, N. Davies, A.. Friday. “An architecture for the effective support of adaptive context-aware applications”. In *Proceedings of the 2nd International Conference in Mobile Data Management* (MDM 2001), pp. 15-26, Hong Kong, 2001. Springer LNCS 1987.

[25] G. Thomson, M. Richmond, S. Terzis, and P. Nixon, “An Approach to Dynamic Context Discovery and Composition”, In *Proceedings of UbiSys '03*, System Support for Ubiquitous Computing Workshop at UbiComp 2003, Seattle, Washington, USA. October 2003.

[26] K. Henricksen, J. Indulska, A. Rakotonirainy, “Modeling Context Information in Pervasive Computing”, In *Proceedings of 1st International Conference Pervasive Comuting* (Pervasive2002), pp. 167-180, 2002. Springer-Verlay LNCS2414.

[27] A. Ranganathan, Roy H. Campbell, **“**A Middleware for Context-Aware Agents in Ubiquitous Computing Environments”, *In ACM/IFIP/USENIX International Middleware Conference 2003*, Rio de Janeiro, Brazil, June 16-20, 2003

[28] M. Roussopoulos, M. Baker, D. Rosenthal, T. Guili, P. Maniatis, J. Mogul, "2 P2P or Not 2 P2P?", In *Proceedings of 3rd International Workshop on Peer-to-Peer Systems*, San Diego, CA, USA, February 26-27, 2004.

[29] J. Risson, T. Moors, "Survey of Research Towards Robust Peer-toPeer Networks: Search Methods", Technical Report UNSW-EE-P2P-1-1, University of South Wales, Sydney, Australia, September 2004.

[30] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks", In *Proceedings of 16th ACM International Conference on Supercomputing* (ICS'02), pp. 84-95, New York, NY, June 2002

[31] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, "Making Gnutella-like P2P Systems Scalable", In *Proceedings of the ACM SIGCOMM 2003*, pp. 407-418, Karlsruhe, Germany, August 2003.

[32] L. A. Adamic, R. M. Lukose, A. R. Puniyani, B. A. Huberman, "Search in Power-Law Networks", In *Physical Review E*, Vol. 64, 2001.

[33] B. Yang, H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks", In *Proceedings of 22nd International Conference on Distributed Computing Systems* (ICDCS 2002), Vienna, Austria, 2002.

[34] A. Crespo, H. Garcia-Molina, "Routing Indices for Peer-to-Peer Systems", In *Proceedings of 22nd International Conference on Distributed Computing Systems* (ICDCS 2002), Vienna, Austria, 2002.

[35] S. C. Rhea, J. Kubiatowicz, “Probabilistic Location and Routing”, In *Proceedings of 21st IEEE INFOCOM 2002*, June 2002.

[36] A. Kumar, J. Xu, E. W. Zegura, “Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks”, In *Proceedings of 24th IEEE INFOCOM 2005*, Miami, US, March 3-17, 2005.

[37] A. Crespo, H. Garcia-Molina, “Semantic Overlay Network for P2P Systems”, Technical Report, Computer Science Department, Stanford University, 2003.

[38] V. Cholvi, P. Felber, E. Biersack, “Efficient Search in Unstructured Peer-to-Peer Networks”, In *European Transaction on Telecommunications*, Special Issues on P2P Networking and P2P Services, Vol. 15, No. 6, November-December 2004

[39] C. Wang, L. Xiao, Y. Liu, P. Zheng, "Distributed Caching and Adaptive Search in Multilayer P2P Networks", In *Proceedings of the 24th International Conference on Distributed Computing Systems* (ICDCS 2004), Tokyo, Japan, March 2004.

[40] H.-C. Hsiao, C.-T. King, S.-Y. Gao. "Making Exploitation of Peers Heterogeneity as a First Class Citizen for Resource Discovery in Peer-to-Peer Networks," In *Proceedings of 1st International Conference on Embedded and Ubiquitous Computing* (EUC'04), pp. 952-961, Aizu, Japan, August 25-27, 2004, LNCS Springer 3207.

[41] S. Jiang, L. Guo, X. Zhang, “LightFlood: an Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems”, In *Proceedings of*

International Conference on Parallel Processing (ICPP'2003), Kaohsiung, Taiwan, ROC, October 6-9, 2003.

[42] K. Sripanidkulchai, B. Maggs, H. Zhang, "Efficient Content Location using Interest-based Locality in Peer-to-Peer Systems, In *Proceedings of 22nd IEEE INFOCOM 2003*, April 2003.

[43] M. Ripeanu, I. Foster, A. Iamnitchi, "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design", in *IEEE Internet Computing Journal special issue on peer-to-peer networking*, Vol. 6, No. 1, 2002.

[44] S. Ratnasamy, M. Handley, R. Karp, S. Shenker, "Topologically-aware Overlay Construction and Server Selection", In *Proceedings of 21st INFOCOM 2002*, pp. 1190-1199, June 2002.

[45] X. Y. Zhang, Q. Shang, Z. Zhang, G. Song, W. Zhu, "A Construction of Locality-Aware Overlay Network: mOverlay and Its Performance", In *IEEE Journal on Selected Area in Communications*, Vol. 22, No. 1, January 2004.

[46] Y. Liu, X. Liu, L. Xiao, L. Ni, X. Zhang, "Location-Aware Topology Matching in P2P Systems", In *Proceedings of 23rd IEEE INFOCOM 2004*, Hong Kong, March 7-11, 2004.

[47] B. Yang, H. Garcia-Molina, "Designing a Super-Peer Network", In *Proceedings of the 19th International Conference on Data Engineering* (ICDE'03), Bangalore, India, March 05-08, 2003.

[48] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", In *ACM SIGCOMM 2001*, San Diego, California, USA, August 27-31, 2001

[49] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, "A Scalable Content Addressable Network", In *ACM SIGCOMM 2001*, San Diego, California, USA, August 27-31, 2001

[50] A. Rowstron, P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large Scale Peer-to-Peer Network", In *Proceedings of 18th IFIP/ACM International Conference on Distributed Systems Platforms* (Middleware 2001), Heidelberg, Germany, November 2001.

[51] T. Berners-Lee, J. Handler, O. Lassila, "The Semantic Web", *Scientific American*, pp. 34-43, May 17, 2001.

[52] C. C. Marshall, F. M. Shipman, "Which Semantic Web", In *Proceedings of 14th ACM Conference on HyperText and HyperMedia*, pp. 57-66, Nottingham, UK, August 26-30, 2003. ACM Press.

[53] O. Lassila, R. Webick, "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation, February 1999. Available at http://www.w3.org/TR/REC-rdf-syntax

[54] D. Brickley, R. V. Guha, "Resource Description Framework (RDF) Schema Specification", W3C Recommendation, March 1999. Available at http://www.w3.org/TR/PR-rdf-schema

[55] T. R. Gruber, “A Translation Approach to Portable Ontologies”, *Knowledge Acquisition*, Vol. 5, No. 2, pp. 199-220, 1993.

[56] OWL Web Ontology Language Reference, W3C Recommendation, February 2004. Available at http://www.w3.org/TR/owl-ref

[57] H. Chen, F. Perich, T. Finin, A. Joshi, "SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications", In *Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services* (Mobiquitous 2004), Boston, MA, August 22-26, 2004.

[58] F. B. Gandon, N. M. Sadeh, “Semantic Web Technologies to Reconcile Privacy and Context Awareness”, *Web Semantics Journal*, Vol. 1, No. 3, November 2003.

[59] R. Masuoka, Y. Labrou, B. Parsia, E. Sirin, “Ontology-Enabled Pervasive Computing Applications”, In *IEEE Intelligent Systems*, Vol. 18, No. 5, pp. 68-72, Sept-Oct, 2003.

[60] F. Perich, A. Joshi, T. Finin, Y. Yesha, “Profile Driven Data Management for Pervasive Environments”, In *13th International Conference on Database and Expert Systems Applications* (DEXA 2002), Aix en Provence, France, September 2002.

[61] G. Chen, D. Kotz, “Context-Sensitive Resource Discovery”, In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pp. 243-252, Fort Worth, Texas, March 2003. IEEE Computer Society Press.

[62] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, J. Lilley, "The design and implementation of an intentional naming system", In *Proceedings of17th ACM SOSP*, Kiawah Island, SC, December 1999.

[63] R. Glassey, G. Stevenson, M. Richmond, F. Wang, P. Nixon, S. Terzis, I. Ferguson, "Towards a middleware for generalised context management." In *1st International Workshop on Middleware for Pervasive and Ad-Hoc Computing* (MPAC03), co-located with Middleware 2003, Rio de Janeiro, Brazil, June 17, 2003.

[64] H.G. Cheng, "Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Sources", doctoral dissertation, Sloan School of Management, MIT, Cambridge, Mass., 1997.

[65] J. Gonzalez-Castillo, D. Trastour, C. Bartolini, "Description logics for Matchmaking Services", HP Laboratories Bristol, Bristol, HPL-2001-265, October 30, 2001.

[66] D. Trastour, C. Bartolini, J. Gonzalez-Castillo, "A Semantic Web Approach to Service Description for Matchmaking of Services", *First International Semantic Web Working Symposium* (SWWS2001), Stanford University, California, USA, pp. 447-462, July 30- August 1, 2001.

[67] A. K. Dey, M. Futakawa, D. Salber, G.D. Abowd, "The Conference Assistant: Combining Context-Awareness with Wearable Computing", In *Proceedings of 3rd International Symposium on Wearable Computing* (ISWC'99), pp. 21-28, San Francisco, CA, USA, October 18-19, 1999.

[68] John Ritter, “Why Gnutella can’t scale. No, really”, February 2001, available at http://www.darkridge.com/~jpr5/doc/gnutella.html

[69] Libby Miller, Andy Seaborne, Alberto Reggiori, “Three Implementations of SquishQL, a Simple RDF Query Language”, In *First International Semantic Web Conference* (ISWC’02), Sardinia, Italy, June 9-12, 2002.

[70] S. Russel, P. Norvig, “Artificial Intelligence: A Modern Approach”, Prentice-Hall, 1995.

[71] Clip2.com Inc., “Bandwidth barriers to Gnutella scalability,” September 2001.

[72] C. R. Palmer, J. G. Steffan, “Generating Network Topologies That Obey Power Laws,” In *IEEE Globecom* 2000, San Francisco, USA, November 27 – December 1, 2000.

[73] T. Strang, C. Linnhoff-Popien, “A Context Modeling Survey”, *Workshop on Advanced Context Modelling, Reasoning and Management*, In the Sixth International Conference on Ubiquitous Computing (Ubicomp2004), Nottingham/England, September 2004.

[74] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. H. Campbell, M. D. Mickunas, “Olympus: A High-Level Programming Model for Pervasive Computing Environments”, In *third annual IEEE International Conference on Pervasive Computing and Communications* (PerCom 2005), Kauai Island, Hawaii, March 8-12, 2005.

[75] R. E. McGrath, A. Ranganathan, R. H. Campbell, M. D. Mickunas, “Incorporating "Semantic Discovery" into Ubiquitous Computing

Infrastructure", In *System Support for Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp 2003)*, Seattle, WA, October 12, 2003

[76] L. Gong, "JXTA: A Network Programming Environment", *IEEE Internet Computing*, Vol. 5, No. 3, pp. 88-95, May-June 2001.

[77] J.J. Carroll, *et al*., "Jena: Implementing the Semantic Web Recommendations", In *Proceedings of 13th International World Wide Web Conference*, pp. 74-83, New York, NY, USA, May 17-22, 2004.

[78] E. Cohen, S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks", In *Proceedings of the ACM SIGCOMM 2002*, Pittsburgh, Pennsylvania, USA, August 19-23, 2003.

[79] C. Gkantsidis, M. Mihail, A. Saberi, "Hybrid Search Schemes for Unstructured Peer-to-Peer Networks", In *Proceedings of 24th IEEE INFOCOM 2005*, Miami, USA, March 13-17, 2005..

[80] L. Kagal, T. Finin, A. Joshi, "A Policy Based Approach to Security for the Semantic Web", In *Proceedings of the 2nd International Semantic Web Conference* (ISWC'03), pp. 402-418, Florida, USA, October 20-23, 2003, Springer-Verlag LNCS2870.

APPENDIX A CoAO ver0.1b XML REPRESENTATION

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:hlt="http://www.i2r.a-star.edu.sg/Orion/htl.owl#/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#/"
    xmlns="http://www.i2r.a-star.edu.sg/Orion/coao.owl#"
  xml:base="http://www.i2r.a-star.edu.sg/Orion/coao.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Network">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="ComputingEntitiy"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ContextDomain"/>
  <owl:Class rdf:ID="Resolution"/>
  <owl:Class rdf:ID="ContextQuality"/>
  <owl:Class rdf:ID="AccessMethod"/>
  <owl:Class rdf:ID="HTTPAccess">
    <rdfs:subClassOf rdf:resource="#AccessMethod"/>
  </owl>
  <owl:Class rdf:ID="Policy"/>
  <owl:Class rdf:ID="ProviderProfile"/>
  <owl:Class rdf:ID="SoftwareProfile">
    <rdfs:subClassOf rdf:resource="#ProviderProfile"/>
  </owl:Class>
  <owl:Class rdf:ID="AdHocActiviy">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Activity"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Device">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#ComputingEntitiy"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ScheduledActivity">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Activity"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="SemWebProf">
    <rdfs:subClassOf rdf:resource="#SoftwareProfile"/>
  </owl:Class>
  <owl:Class rdf:ID="Agent">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#ComputingEntitiy"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="WebServiceProf">
    <rdfs:subClassOf rdf:resource="#SoftwareProfile"/>
  </owl:Class>
  <owl:Class rdf:ID="Application">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#ComputingEntitiy"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Location">
    <rdfs:subClassOf rdf:resource="#ContextDomain"/>
  </owl:Class>
  <owl:Class rdf:ID="SensorProfile">
    <rdfs:subClassOf rdf:resource="#ProviderProfile"/>
  </owl:Class>
  <owl:Class rdf:ID="Validity"/>
  <owl:Class rdf:ID="Context"/>
  <owl:Class rdf:ID="ImageSen">
```

```
  <rdfs:subClassOf rdf:resource="#SensorProfile"/>
</owl:Class>
<owl:Class rdf:ID="User">
  <rdfs:subClassOf rdf:resource="#ContextDomain"/>
</owl:Class>
<owl:Class rdf:ID="OutdoorLocation">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:ID="ContextProvider"/>
<owl:Class rdf:ID="AccessModel"/>
<owl:Class rdf:ID="IndoorLocation">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>
<owl:Class rdf:ID="Time"/>
<owl:Class rdf:ID="Hour"/>
<owl:Class rdf:ID="Minute"/>
<owl:Class rdf:ID="Second"/>
<owl:Class rdf:ID="Date"/>
<owl:Class rdf:ID="Service">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ComputingEntitiy"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#ComputingEntitiy">
  <rdfs:subClassOf rdf:resource="#ContextDomain"/>
</owl:Class>
<owl:Class rdf:about="#Activity">
  <rdfs:subClassOf rdf:resource="#ContextDomain"/>
</owl:Class>
<owl:Class rdf:ID="KBaseProf">
  <rdfs:subClassOf rdf:resource="#SoftwareProfile"/>
</owl:Class>
<owl:Class rdf:ID="AudioSen">
  <rdfs:subClassOf rdf:resource="#SensorProfile"/>
</owl:Class>
<owl:Class rdf:ID="MotionSen">
  <rdfs:subClassOf rdf:resource="#SensorProfile"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="accessModel">
  <rdfs:domain rdf:resource="#ContextProvider"/>
  <rdfs:range rdf:resource="#AccessModel"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="provides">
  <rdfs:domain rdf:resource="#ContextProvider"/>
  <rdfs:range rdf:resource="#Context"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="accessMethod">
  <rdfs:range rdf:resource="#AccessMethod"/>
  <rdfs:domain rdf:resource="#AccessModel"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="time">
  <rdfs:range rdf:resource="xsd:time"/>
  <rdfs:domain rdf:resource="#Time"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hour">
  <rdfs:domain rdf:resource="#Time"/>
  <rdfs:range rdf:resource="#Hour"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="minute">
  <rdfs:domain rdf:resource="#Time"/>
  <rdfs:range rdf:resource="#Minute"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="second">
  <rdfs:domain rdf:resource="#Time"/>
  <rdfs:range rdf:resource="#Second"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID=”hasHour”>
  <rdfs:domain rdf:resource=”#Hour/>
  <rdfs:range>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="23"/>
    </xsd:restriction>
  </rdfs:range>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID=”hasMinute”>
```

```
  <rdfs:domain rdf:resource=”#Minute/>
  <rdfs:range>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="59"/>
    </xsd:restriction>
  </rdfs:range>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID=”hasSecond”>
  <rdfs:domain rdf:resource=”#Second>
  <rdfs:range>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="59"/>
    </xsd:restriction>
  </rdfs:range>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="giveRightTo">
  <rdfs:domain rdf:resource="#Policy"/>
  <rdfs:range rdf:resource="#User"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasQuality">
  <rdfs:domain rdf:resource="#Context"/>
  <rdfs:range rdf:resource="#ContextQuality"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="resolution">
  <rdfs:range rdf:resource="#Resolution"/>
  <rdfs:domain rdf:resource="#ContextQuality"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="policy">
  <rdfs:range rdf:resource="#Policy"/>
  <rdfs:domain rdf:resource="#AccessModel"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasProfile">
  <rdfs:range rdf:resource="#ProviderProfile"/>
  <rdfs:domain rdf:resource="#ContextProvider"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="valid">
  <rdfs:range rdf:resource="#Validity"/>
  <rdfs:domain rdf:resource="#ContextQuality"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasDomain">
  <rdfs:range rdf:resource="#ContextDomain"/>
  <rdfs:domain rdf:resource="#Context"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="samplingRate">
  <rdfs:range rdf:resource="xsd:float"/>
  <rdfs:domain rdf:resource="#SensorProfile"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="samplingUnit">
  <rdfs:range rdf:resource="#Time"/>
  <rdfs:domain rdf:resource="#SensorProfile"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="ValidFrom">
  <rdfs:domain rdf:resource="#Validity"/>
  <rdfs:range rdf:resource="xsd:dateTime"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ValidUntil">
  <rdfs:domain rdf:resource="#Validity"/>
  <rdfs:range rdf:resource="xsd:dateTime"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ValidPeriod">
  <rdfs:range rdf:resource="xsd:int"/>
  <rdfs:domain rdf:resource="#Validity"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="date">
  <rdfs:range rdf:resource="xsd:string"/>
  <rdfs:domain rdf:resource="#Validity"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="vendor">
  <rdfs:domain rdf:resource="#ProviderProfile"/>
  <rdfs:range rdf:resource="xsd:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="lengthPrecision">
  <rdfs:domain rdf:resource="#Resolution"/>
  <rdfs:range rdf:resource="xsd:string"/>
```

```
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="weightUnit">
  <rdfs:range rdf:resource="xsd:string"/>
  <rdfs:domain rdf:resource="#Resolution"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serviceDomain">
  <rdfs:domain rdf:resource="#WebServiceProf"/>
  <rdfs:range rdf:resource="xsd:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Protocol">
  <rdfs:domain rdf:resource="#AccessMethod"/>
  <rdfs:range><owl:DataRange><owl:oneOf>
     <rdfs:List>
        <rdf:first rdf:datatype="&xsd;string">HTTP</rdf:first>
        <rdf:rest>
          <rdfs:List>
            <rdf:first rdf:datatype="&xsd;string">SOAP</rdf:first>
            <rdf:rest>
              <rdf:List>
                 <rdf:first rdf:datatype="&xsd;string">FTP</rdf:first>
                 <rdf:rest rdf:resource="&rdf;nil"/>
              </rdf:List></rdf:rest>
          </rdf:List></rdf:rest>
     </rdf:List></rdf:rest>
   </owl:oneOf></owl:DataRange></rdf:range>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ip">
  <rdfs:range rdf:resource="xsd:string"/>
  <rdfs:domain rdf:resource="#AccessMethod"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="to">
  <rdfs:range rdf:resource="xsd:int"/>
  <rdfs:domain rdf:resource="#Validity"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="channel">
  <rdfs:range rdf:resource="xsd:string"/>
  <rdfs:domain rdf:resource="#AccessModel"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="weightPrecision">
  <rdfs:domain rdf:resource="#Resolution"/>
  <rdfs:range rdf:resource="xsd:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="sensorType">
  <rdfs:domain rdf:resource="#ProviderProfile"/>
  <rdfs:range rdf:resource="xsd:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="samplingRate">
  <rdfs:range rdf:resource="xsd:float"/>
  <rdfs:domain rdf:resource="#ProviderProfile"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="lengthUnit">
  <rdfs:domain rdf:resource="#Resolution"/>
  <rdfs:range rdf:resource="xsd:string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="physicalLocation">
  <rdfs:range rdf:resource="xsd:string"/>
  <rdfs:domain rdf:resource="#ProviderProfile"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="HTTPURL">
  <rdfs:domain rdf:resource="#HTTPAccess"/>
  <rdfs:range rdf:resource="xsd:anyURI"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="PortNumber">
  <rdfs:domain rdf:resource="#HTTPAccess"/>
  <rdfs:range>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="1024"/>
      <xsd:maxInclusive value="65535"/>
    </xsd:restriction>
  </rdfs:range>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="providerID">
  <rdfs:domain rdf:resource="#ContextProvider"/>
  <rdfs:range rdf:resource="xsd:anyURI"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="spaceLocation">
```

```
    <rdfs:domain rdf:resource="#ContextProvider"/>
    <rdfs:range rdf:resource="&hlt;Singapore"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="port">
    <rdfs:range rdf:resource="xsd:int"/>
    <rdfs:domain rdf:resource="#AccessMethod"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="sampleRate">
    <rdfs:domain rdf:resource="#Resolution"/>
    <rdfs:range rdf:resource="xsd:float"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="reasoningTechnique">
    <rdfs:domain rdf:resource="#KBaseProf"/>
    <rdfs:range rdf:resource="xsd:string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="timeUnit">
    <rdfs:domain rdf:resource="#Resolution"/>
    <rdfs:range rdf:resource="xsd:string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="correctness">
    <rdfs:range>
      <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="0.0"/>
        <xsd:maxInclusive value="1.0"/>
      </xsd:restriction>
    </rdfs:range>
    <rdfs:domain rdf:resource="#ContextQuality"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="capacity">
    <rdfs:range rdf:resource="xsd:int"/>
    <rdfs:domain rdf:resource="#ContextQuality"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="cost">
    <rdfs:range rdf:resource="xsd:float"/>
    <rdfs:domain rdf:resource="#ContextQuality"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

APPENDIX B RDQL GRAMMAR*

```
CompilationUnit          ::=   Query <EOF>
CommaOpt                 ::=   ( <COMMA> )?
Query                    ::=   SelectClause ( SourceClause )? TriplePatternClause
                               ( ConstraintClause )? ( PrefixesClause )?
SelectClause             ::=   ( <SELECT> Var ( CommaOpt Var )* | <SELECT> <STAR> )
SourceClause             ::=   ( <SOURCE> | <FROM> ) SourceSelector ( CommaOpt
                               SourceSelector )*
SourceSelector           ::=   QName
TriplePattern            ::=   <WHERE> TriplePattern ( CommaOpt TriplePattern )*
Clause
ConstraintClause         ::=   <SUCHTHAT> Expression ( ( <COMMA> | <SUCHTHAT> )
                               Expression )*
TriplePattern            ::=   <LPAREN> VarOrURI CommaOpt VarOrURI CommaOpt
                               VarOrConst <RPAREN>
VarOrURI                 ::=   Var
                         |     URI
VarOrConst               ::=   Var
                         |     Const
Var                      ::=   "?" Identifier
PrefixesClause           ::=   <PREFIXES> PrefixDecl ( CommaOpt PrefixDecl )*
PrefixDecl               ::=   Identifier <FOR> <QuotedURI>
Expression               ::=   ConditionalOrExpression
ConditionalOr            ::=   ConditionalAndExpression ( <SC_OR>
Expression                     ConditionalAndExpression )*
ConditionalAnd           ::=   StringEqualityExpression ( <SC_AND>
Expression                     StringEqualityExpression )*
StringEquality           ::=   ArithmeticCondition ( <STR_EQ> ArithmeticCondition |
Expression                     <STR_NE> ArithmeticCondition | <STR_MATCH>
                               PatternLiteral | <STR_NMATCH> PatternLiteral )*
Arithmetic               ::=   EqualityExpression
Condition
Equality                 ::=   RelationalExpression ( <EQ> RelationalExpression |
Expression                     <NEQ> RelationalExpression )?
Relational               ::=   AdditiveExpression ( <LT> AdditiveExpression | <GT>
Expression                     AdditiveExpression | <LE> AdditiveExpression | <GE>
                               AdditiveExpression )?
Additive                 ::=   MultiplicativeExpression ( <PLUS>
Expression                     MultiplicativeExpression | <MINUS>
                               MultiplicativeExpression )*
Multiplicative           ::=   UnaryExpression ( <STAR> UnaryExpression | <SLASH>
Expression                     UnaryExpression | <REM> UnaryExpression )*
UnaryExpression          ::=   UnaryExpressionNotPlusMinus
                         |     ( <PLUS> UnaryExpression | <MINUS> UnaryExpression )
UnaryExpression          ::=   ( <TILDE> | <BANG> ) UnaryExpression
NotPlusMinus
                         |     PrimaryExpression
PrimaryExpression        ::=   Var
                         |     Const
                         |     <LPAREN> Expression <RPAREN>
Const                    ::=   URI
                         |     NumericLiteral
                         |     TextLiteral
                         |     BooleanLiteral
                         |     NullLiteral
```

* Available at “http://jena.sourceforge.net/RDQL/rdql_grammar.html”

```
NumericLiteral      ::=   ( <INTEGER_LITERAL> | <FLOATING_POINT_LITERAL> )
TextLiteral         ::=   ( <STRING_LITERAL1> | <STRING_LITERAL2> ) ( <AT>
                          Identifier )? ( <DATATYPE> URI )?
PatternLiteral      ::=
BooleanLiteral      ::=   <BOOLEAN_LITERAL>
NullLiteral         ::=   <NULL_LITERAL>
URI                 ::=   <QuotedURI>
                    |     QName
QName               ::=   <NSPrefix> ':' (<LocalPart>)?
                          Unlilke XML Namespaces, the local part is optional
Identifier          ::=   ( <IDENTIFIER> | <SELECT> | <SOURCE> | <FROM> |
                          <WHERE> | <PREFIXES> | <FOR> | <STR_EQ> | <STR_NE> )
```