2809342305

EFERENCE ONLY

UNIVERSITY OF LONDON THESIS

egree PhD Year 2007 Name of Author ROEL MAGLENTE OCAMPO

OPYRIGHT

is is a thesis accepted for a Higher Degree of the University of London. It is an published typescript and the copyright is held by the author. All persons consulting e thesis must read and abide by the Copyright Declaration below.

COPYRIGHT DECLARATION

ecognise that the copyright of the above-described thesis rests with the author and at no quotation from it or information derived from it may be published without the ior written consent of the author.

OAN

heses may not be lent to individuals, but the University Library may lend a copy to pproved libraries within the United Kingdom, for consultation solely on the premises those libraries. Application should be made to: The Theses Section, University of ondon Library, Senate House, Malet Street, London WC1E 7HU.

EPRODUCTION

niversity of London theses may not be reproduced without explicit written ermission from the University of London Library. Enquiries should be addressed to e Theses Section of the Library. Regulations concerning reproduction vary ccording to the date of acceptance of the thesis and are listed below as guidelines.

. Before 1962. Permission granted only upon the prior written consent of the author. (The University Library will provide addresses where possible).

. 1962 - 1974. In many cases the author has agreed to permit copying upon completion of a Copyright Declaration.

. 1975 - 1988. Most theses may be copied upon completion of a Copyright Declaration.

). 1989 onwards. Most theses may be copied.

This thesis comes within category D.

☐ This copy has been deposited in the Library of UCL

☐ This copy has been deposited in the University of London Library, Senate House, Malet Street, London WC1E 7HU.

Department of Electronic and Electrical Engineering
University College London
University of London

Understanding, Modeling and Using Flow Context

Roel Maglente Ocampo

Submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
at the University of London

December 2006

UMI Number: U593070

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI U593070
Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Declaration

I, Roel M. Ocampo, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Roel M. Ocampo

Abstract

This thesis presents a concept called flow context, defined as any information that can be used to characterize the situation of a sequence of protocol data units, called a flow, within a network. Flow context is designed to enable the realization of context-aware networks: networks that can sense, process, disseminate and use context information in order to enable or trigger services, or modify and optimize their operation.

The thesis discusses a conceptualization for flow context, and describes its characteristics. A semantic model for flows and flow context in the form of an OWL-DL ontology is developed. The various components of the flow context life cycle, consisting of the stages of sensing, processing, dissemination and use, are described.

To account for its multi-faceted nature, a multi-dimensional approach to sensing flow context is adopted. Novel implementations for three exemplar sensors, including sensors for intrinsic flow context, node and device characteristics, and for device location, are presented.

Mechanisms for locating flow context sensors and propagating context event notifications using a distributed hash table (DHT) are described and evaluated. Simulation results suggest that DHTs can provide decentralized and scalable solutions for flow context location and dissemination. In addition, a novel mode of context distribution called path-coupled flow context dissemination is described.

An evaluation of semantic flow context processing using the RacerPro reasoner is presented. Various platform-specific reasoning modes are tested, including the use of queries, ABox modification, and incomplete reasoning.

Finally, several application scenarios illustrating the potential uses of flow context in areas such as mobility and moving networks, quality of service, intelligent flow classification, network management, and other applications, are presented. Many of these scenarios are demonstrated through proof-of-concept implementations, which may be further evaluated and developed into full, working, and useful applications.

Acknowledgements

First, I would like to thank my supervisors: Professor Chris Todd, for all the support and guidance, and for the thorough review of this thesis, which I truly appreciate; Professor Hermann de Meer, for ushering me into the world of postgraduate research and supervising me during my first two years at UCL; and Dr. Miguel Rio, for some thought-provoking questions during my transfer viva that made me think more about my work. My examiners, Professor David Hutchison of Lancaster University, and Dr. Stefan Poslad of Queen Mary University of London, also made numerous helpful comments that have greatly improved the presentation of this document. Any errors or omissions that remain, of course, are my own.

Professor Alex Galis generously supported my work and enabled me to participate in collaborative EU research, particularly the Ambient Networks Project. I have benefited much from the interactions with the Ambient Networks project partners, particularly from the constructive comments and feedback from Mikhail Smirnov and Chris Reichert of Fraunhofer FOKUS regarding some of the concepts described in this work.

Racer Systems GmBH was kind enough to provide a free academic license for the use of RacerPro.

My postgraduate study at UCL was supported by a Doctoral Study Fellowship from the University of the Philippines System and counterpart funding from the University of the Philippines Diliman, and I would like to thank the university officials and staff who made this possible. My friends and colleagues at the UP Electrical and Electronics Engineering and the Computer Science departments continuously provided valuable advice and moral support from halfway around the world – email, instant messaging and SMS are truly wonderful technologies!

I would like to thank the people at GS104 for the stimulating exchanges and discussions, for sharing interesting ideas and giving constructive feedback, and for making my stay at the lab a pleasant and entertaining experience. It had been a privilege working with you.

Finally, and most of all, I would like to thank my family. Jane, Pauline, and Lean: this work is dedicated to you.

Contents

List of Figures

List of Abbreviations

AN	Ambient networks
CAN	Content Addressable Networks
CC/PP	Composite Capability/Preferences Profiles
ConCoord	Context Coordinator
DHT	Distributed hash table
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IPFIX	IP Flow Information Export
MIB	Management Information Base
MN	Mobile node
OWL	Web Ontology Language
QoC	Quality of context
QoS	Quality of service
RDF	Resource Description Framework
SNMP	Simple Network Management Protocol
UAProf	User Agent Profile
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Chapter 1

Introduction

Today we find ourselves almost completely blanketed by a plethora of wired and wireless overlay networks [1] including those provided by different mobile phone services, privately-owned and public "hotspot" wireless LANs, personal area networks that use Bluetooth, as well as satellite-based mobile communication and Internet services. The diversity of these networks is only rivalled by variety of the features and characteristics of the mobile devices used on them, and the applications that run on these devices and networks.

As we shift from one activity to the next, we find it increasingly inconvenient, if not outright difficult, to continuously and consciously adapt to the different devices and connectivity modes appropriate to our activities, as well as to the constant changes in network characteristics and conditions. Such a situation would simply demand too much of our attention. As we move towards having ubiquitous network connectivity, we face the challenge of transforming networks into "invisible" [2] tools, much in the same way that Mark Weiser identified the challenge of attaining invisibility in the area of Ubiquitous Computing:

> "A good tool is an invisible tool. By invisible, I mean that the tool does not intrude on your consciousness; you focus on the task, not the tool. Eyeglasses are a good tool – you look at the world, not the eyeglasses. The blind man tapping the cane feels the street, not the cane. Of course, tools are not invisible in themselves, but part of a context of use. With enough practice we can make many apparently difficult things disappear: my fingers know vi editing commands that my conscious mind has long forgotten. But good tools enhance invisibility" [3].

So how does the network become a useful yet invisible, or more reasonably, a *minimally-distracting* [4] tool? A network that requires little or no attention from the user might have the following characteristics:

1. *Little or no user configuration.* The network requires little or no explicit configuration nor re-configuration by the user even when the user's goal, task or environment changes. The term "environment" here includes the computing and access devices used, the network, applications, other users, or any other objects or factors that influence the user's task. There may be cases however where the user might actually prefer to exercise some form of control over configuration choices; in these cases the user should be provided the opportunity to make such decisions in an informed manner.

2. *Quality of service guarantees and adaptive capabilities.* The network is able to provide the services necessary for the user to achieve her goals, and is able to offer a performance level that strikes a balance between the user's task at hand, the level of available resources, and the needs of other users. It should be able to adapt in the face of changes in the user's activities or in his computing and network environment in order to maintain this delicate balance. This is also called self-tuning [4, 5].

3. *Reliability and availability.* The network and its services should provide certain levels of reliability, availability, and fault-tolerance, perhaps commensurate to the importance of the task at hand.

4. *Security and privacy.* The network should provide a certain degree of privacy and security appropriate to the user's task, or based on the user's preferences, or at a level that the user is comfortable with.

In networked environments, where resources such as bandwidth are often shared, it may not be feasible for the network to simultaneously satisfy the needs of all individual users under all possible situations. A more pragmatic approach is to set an adaptation policy P that applies to class of users U, for a particular type of usage or activity A, under a certain situation or set of network conditions C. In other words, an adaptation P is triggered by the general parameters (U, A, C). An important requirement, therefore is to determine whether (1) the condition C exists within the network or user environment, (2) the traffic belongs to the class of users U, and (3) if the traffic is the result of usage activity A. It may be said that these parameters describe a certain situation or *context* within the network.

1.1 Aims and motivation

The work in this thesis is motivated by the fact that prior work on adaptive networking, as will be seen in the next chapter, did not have a comprehensive framework and architectural support for obtaining, processing, distributing and using the wide range of information now known as *context* in the literature. As a result these previous efforts failed to enable the wide range of optimizations and adaptation required in today's highly dynamic and heterogeneous networking environments. This thesis aims to enable these by providing a framework for the management and utilization of context information within networks.

Context is defined in the New Oxford Dictionary of English as "the circumstances that form the setting for an event ... and in terms of which it can be fully understood and assessed." An often-cited definition of context in the human-computer interaction (HCI) and pervasive computing domains comes from the work of Dey, Salber and Abowd, who define context as "any information that can be used to characterize the situation of entities ... that are considered relevant to the interaction between a user and an application, including the user and the applications themselves" [6]. Perhaps due to its origins, the interest and work in context has largely revolved around the use of the situations of users, i.e., "user-centric context" [7], in triggering execution or adaptation in applications. However, as this thesis is concerned with the interactions between users and applications via the network, or more directly, on the interaction between users and networks, it focuses more on the use of context information by the network rather than by applications. In other words, rather than context-aware applications, the focus and main motivation behind this work is on building *context-aware networks* through the application and use of context-awareness within the network domain.

In order to advance the study of context-aware networks, the work in this thesis aims to further develop its conceptual foundations, contribute to the design and implementation of architectural components, and explore its potential applications and implications. Specifically, in order to enable the systematic use of context information within networks, the work in this thesis aims to:

1. Define and develop concepts that may be useful in understanding, designing, implementing, operating and managing context-aware networks;
2. Identify specific entities within the network whose context could be sensed, processed and used;
3. Develop models for the context information describing these network entities;

4. Develop architectural and software components that would allow such context information to be obtained, processed, distributed and used within the network and end-applications; and

5. Investigate illustrative application scenarios for the use of context information within networks.

Through the use of context information, networks would be able to achieve some, if not most of the desired characteristics previously mentioned, such as the ability to dynamically adapt, automatically reconfigure, and provide quality of service, privacy and security assurances. The approach advocated here is in agreement with Dobson, who writes:

> "(we) advocate instead increasing the amount of information available to a network sub-system about the content it is carrying and the context in which that content will be used. We argue that this view of the network as an equal partner in interactions – rather than as a simple packet carrier – is an appropriate reaction to the desire for autonomic communication systems that facilitates a range of optimisations currently difficult to accomplish in a scalable fashion" [8]

The apparent lack of a solid conceptual foundation for context-aware networks also serves as a specific motivation for the work in this thesis. While the concepts used in the domain of context-aware *applications* (such as in the fields of Ubiquitous Computing and Human-Computer Interaction) are still under continuous development, relatively much progress has been made there. Recently however there have been attempts to import the concepts developed in those fields – particularly the definition of the term "context" – into the domain of computer networks in order to define concepts such as "network-centric context" [7] and "network-related context" [9]. While these concepts have appeared in the literature perhaps as an attempt to draw a distinction from the "user-centric" type of context used in the HCI domain, these abstract definitions still have to be complemented by more concrete, operational concepts that can easily be translated into actual realizations of context-aware networks. Thus, Section 3.1.2 discusses why "network-related context" or "network-centric context" seem to be poorly-defined, or at the very least, not yet well-understood. Section 3.2 then explains how a better understanding of the nature and use of context within the network can be achieved by focusing on the context of specific entities, such as *flows* within the network, rather than on a nebulous concept of "network context."

Once the contextual entities of interest have been identified, it is also necessary to develop appropriate representations and models for their context information, so that the information may be obtained, processed, shared, and used by other entities. The work here

specifically aims to investigate the applicability and use of *ontologies* to model context information within the network, enabling context to be represented using formal and standardized languages, and allowing them to be semantically processed and reasoned upon.

The development of architectural and software components that would be useful for the design and implementation of actual context-aware networks is another objective of the work in this thesis. The aim is to develop and present novel designs, mechanisms and implementations of components that help sense, collect, aggregate and distribute context information within context-aware networks.

1.2 Thesis structure

In accordance with these research objectives, the rest of this thesis is organized as follows:

Chapter 2 ("From Adaptive to Context-Aware Networks") sets the scene for the work in this thesis by reviewing related work in context-aware application architectures, adaptive networks, and the recent work in context-aware networks. Previous efforts in adaptive networking are analyzed to gain insights into the types of information used for adaptation and the methods employed to use it. The review of context-aware *applications* on the other hand examines the distinction between adaptive and context-aware networks, and identifies the functional components needed to make networks "context-aware." The ongoing efforts in the development of context-aware networks are then examined to position the work described in this thesis within a broader perspective.

Chapter 3 ("Understanding Flow Context: Concepts") establishes the conceptual foundations of this thesis. The concepts of context and context-awareness are defined, and an examination is made on how these concepts were imported into the field of computer networks to arrive at a notion of "network context." An alternative approach that focuses on flows as entities of interest within the network, using the concept of *flow context* is proposed. A detailed definition and description of the characteristics and *life cycle* of flow context are provided.

Chapter 4 ("Modeling Flow Context") discusses a formal model for flows and their context based on *ontologies*. This chapter attempts to answer the question: "How do we represent flow context?" The merits of an ontology-based approach to flow context modeling are presented, followed by a detailed discussion on implementation of an ontology for flows and their context. Since the ontology uses technologies developed for the Semantic Web, an overview of these technologies is provided in Appendix D.

Having identified the types and characteristics of flow context in previous chapters, *Chapter 5* ("Sensing Flow Context"), answers the question: "How do we obtain flow context?" The notion of using *context sensors* to perform this task is introduced, and the design and implementation of some representative sensors, covering a variety of types of flow context, is described.

Chapter 6 ("Aggregating and Disseminating Flow Context") discusses the functions that bridge sensing and using flow context: the discovery and location of context sources, the processing and aggregation of the information, and its subsequent dissemination within the network. A method for locating and disseminating flow context using distributed hash tables (DHTs) and a simulator developed for its evaluation are presented. The semantic processing of flow context using a reasoner is also evaluated in detail.

In *Chapter 7* ("Using Flow Context: Applications") some of the potential applications of flow context are explored. The use of flow context in implicit QoS signaling, intelligent flow classification and management, and in host mobility and moving networks are demonstrated using proof-of-concept demonstrations. Other potential applications, such as in overlay routing, content delivery, attack mitigation and control, network security, privacy, accounting, billing, and network management are also explored.

Chapter 8 ("Conclusion and Future Work") summarizes and concludes this thesis, discusses some open questions, and outlines future work.

Figure 1.1 illustrates the organization of key topics in this thesis, mapping their location and links with related sections.

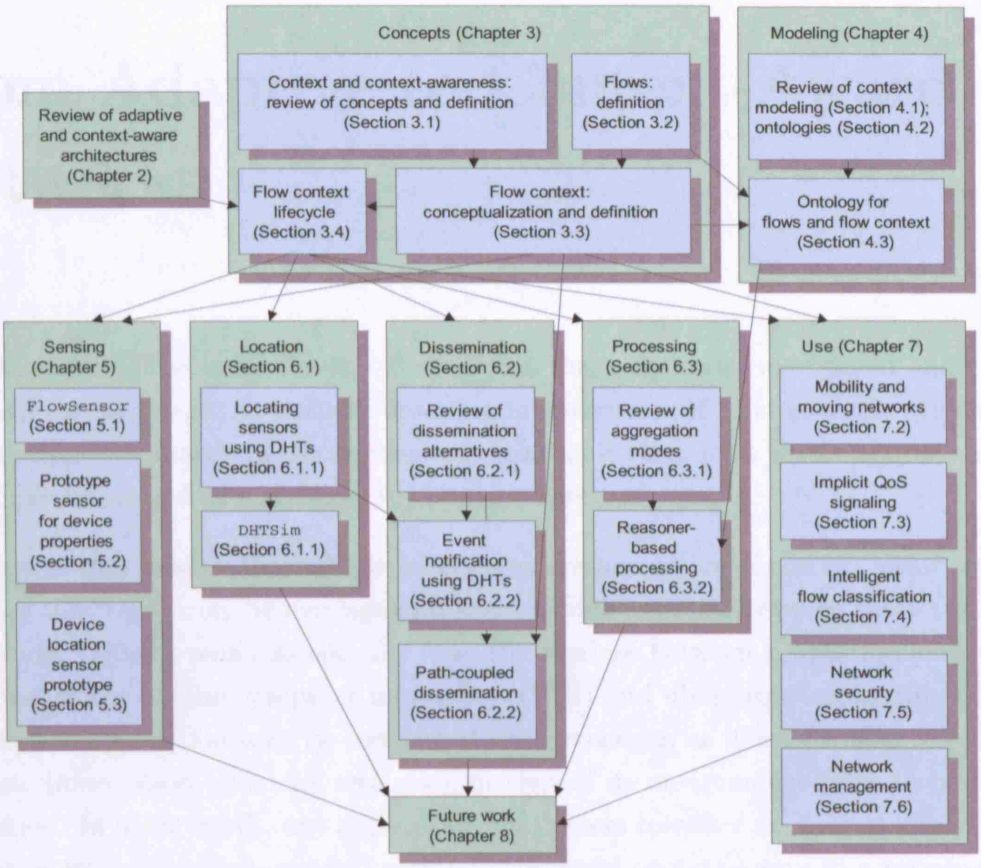


Figure 1.1: Organization of this thesis, showing the location of key topics and their links with related sections.

Chapter 2

From Adaptive to Context-Aware Networks

This chapter sets the scene for the work in this thesis by reviewing related and relevant prior and ongoing work. It initially describes architectures of context-aware systems that were developed primarily in the application domain. It then deals with *adaptive networks*, which may be viewed as precursors to *context-aware networks*.

The view in this thesis – that adaptive networks are precursors of context-aware networks – implies that there may be overlaps and shared characteristics between these two classes of networks. This be seen conceptually from the overlaps between adaptation and context-awareness in the human-computer interaction (HCI) and ubiquitous computing domains. An adaptive application may be considered context-aware, as it would have to sense and use some information about its own state or that of its environment prior to performing adaptation. In other words, any application that reacts to either explicit or implicit input (the latter from its sensed context) may be considered context-aware to a certain extent, given the broad definition of the term "context" [6].

On the other hand, although many applications that are context-aware are adaptive, not all context-aware applications perform automatic adaptation in response to context. For example, applications that collect and process context in order to present relevant information to the user, thereby allowing the *user* (rather than the application) to adapt or select the next course of action accordingly, may also be considered context-aware [10]. In other words, it is possible to have a context-aware application that does not automatically perform adaptation.

The same overlaps and distinctions are used in this thesis between adaptive and context-aware networks. Adaptive networks may be considered context-aware, however, most of the *early* adaptive networks lacked the formal architecture and a comprehensive set of components for the systematic collection, processing, aggregation and distribution of context information – architectural components that may be considered hallmarks of context-aware networks. On the other hand, not all context-aware networks might be adaptive. It is conceivable to build a context-aware network whose main context-related task would be to sense and process context information, not for its own use, but for the use of other entities such as end-applications, or even by humans.

The rest of this chapter is organized as follows: in order to identify the architectural characteristics that distinguish the early adaptive networks from the emerging context-aware ones, Section 2.1 surveys some representative context-aware applications, focusing on their functional and architectural components. This is followed by a survey of early efforts in building adaptive networks, in Section 2.2. Section 2.3 then provides an overview of recent efforts to integrate context-related architectural components into networks, leading to the emergence of context-aware networks.

2.1 Context-aware architectures

In order to specify and design the architectural components that would support the functions of context sensing, distribution, processing and aggregation of context within networks, it is necessary to review the work done on context-awareness in the application domain. However, rather than present another survey of context-aware applications and architectures, only the essential features and components that describe how context is obtained and processed in general are summarized and presented here. Interested readers may instead refer to [10, 11, 12, 13] for more general surveys on context-aware applications.

Pascoe identified four generic capabilities needed by context-aware applications, namely contextual sensing, adaptation, resource discovery, and augmentation [14]. In Pascoe's framework, contextual sensing referred to the detection of environmental states and their subsequent presentation to the user. Applications then adapted their behavior to this contextual knowledge. Contextual augmentation extended these capabilities further by adding information, either through the digital data augmenting reality, or reality augmenting digital data. Contextual resource discovery made context information and information resources available to interested entities.

Several context-aware architectures evolved from work on location systems. Wallbaum proposed a layered system model for indoor geolocation systems that included data collection, location computation, location normalization, and location provisioning components [15]. In contrast with Pascoe's approach, Wallbaum decoupled location sensing into separate data collection and computation components, permitting the use of different positioning algorithms. Normalization then transformed the computed location into a standardized representation, and the information was provided to applications in the provisioning step. Hightower, Brumitt and Borriello introduced the Location Stack model which similarly focused on location context, and consisted of a seven-layer stack that included sensors, measurements, fusion, arrangements, contextual fusion, activities and intentions [16]. Although the models introduced by Wallbaum and Hightower et al. were originally proposed to handle location context, the essential components may be applied to other types of context sensing.

Dey, Salber and Abowd offered a conceptual framework that included context widgets, interpreters, aggregators, services and discoverers [6]. Context widgets abstracted underlying sensors and acquired context information. This information was further abstracted by interpreters into higher-level information. Aggregators gathered information relevant to an entity. Services then executed behaviors using context. Discoverers maintained information on which of these components are available for use by applications.

Schmidt proposed a "perception architecture for context-aware systems," consisting of sensors, cues, contexts, and the applications that used them [11]. In this architecture, physical and logical sensors provided information about the world, to be abstracted or processed into symbolic or sub-symbolic values called cues. The context layer then abstracted cues into situations and decided whether a situation satisfied the definition of a particular context. The context was then passed on to applications.

Finally, Ocampo and de Meer suggested a framework for "cognitive services" that provided functions for context sensing, interpretation, augmentation and adaptation by *networks*, rather than by applications [17]. Sensing referred to the collection of measurable or quantifiable physical data or the observation of an event. Interpretation involved the application of a numerical process, an algorithm, or a logical process such as inference or reasoning to transform the raw data into a more relevant or useful form, or to detect the occurrence of an event of interest. Augmentation involved the examination or aggregation of recently-sensed or interpreted events and data with other pieces of information, such as data or events originating from other sensors (sensor fusion), or historical data from the same sensor. Adaptation consisted of the specific actions in response to the detection of certain contexts, or changes in these contexts, including service triggering, modification of

application or network behavior, or learning.

2.2 Network adaptation

In this section, network architectures that demonstrated adaptivity in the face of changes in network conditions or user requirements are discussed. Although they used some forms of context within the network, in general, they only provided implicit and limited support for context. Thus, these architectures should be considered as *precursors* to the emergence of context-aware networks. Nonetheless, these architectures still provide some important lessons, to be summarized and presented in Section 3.4.1.

The early work on network adaptation may be traced back to the emergence in the early 1990s of heterogenous multimedia-enabled devices and applications that required networks to transport graphics, digital audio, and digital video, in addition to text and other forms of data. To deal with transporting these types of traffic, for instance, Campbell, Coulson and Hutchison developed an integrated, multilayer Quality of Service Architecture (QoS-A) spanning end-systems and the network [18]. QoS-A allowed the required performance parameters of media flows in an asynchronous transfer mode (ATM) network to be expressed in a service contract. Fluctuations in the network service, resulting in violations to the service contract could be signalled back to the user, allowing the latter to adapt accordingly. Such adaptation might include readjusting the application's performance to accommodate current network conditions, re-negotiation of the flow's QoS, disconnection from the service, or no action. Subsequent work by Yeadon, Garcia, Campbell and Hutchison, likewise dealing with continuous media flows in heterogeneous environments, focused on the issue of *flow management*, particularly addressing the areas of QoS adaptation and flow filtering [19]. A flow management model incorporated three main QoS policies, including a resource selection policy for resource allocation and sharing, a media conversion policy for media encoding conversion and scaling, and a QoS adaptation policy to alter encoding characteristics at the source. Sources and receivers then interacted with a flow manager, which was responsible for setting up and maintaining communication group membership and resource sharing lists, flow set-up, resource reservation, routing, and filter management.

The early adaptive architectures represented by the work of Campbell et al. and Yeadon et al. focused solely on providing QoS-related adaptive mechanisms that allowed the network to deal with multimedia flows, and thus the context information of interest, primarily involved traffic parameters and knowledge of the media type carried within the flow. Subsequent work on adaptive networks, as will be shown in this section, enabled other types

of adaptation such as reassembly, buffering, encryption, and adaptation to mobility, in addition to QoS-related adaptation. Although the body of work on adaptive networks is quite extensive, a sampling of representative works is presented here. These were selected to provide a view of the different approaches in designing adaptive networks, including:

- the different levels of end-application involvement in the adaptation process, e.g. whether adaptation is "application-aware" (Odyssey), or if it only involves the network (Conductor);
- where the adaptation may take place, such as within end-hosts (Odyssey), within the network edge (TranSend), within specific network segments (Transformer Tunnels), or in a distributed fashion within the network (Conductor);
- what type of context information is sensed, such as traffic parameters (Congestion Manager) or flow content type (Transend).

A brief discussion of these works is presented in the next few sections.

2.2.1 TranSend

Fox et al. introduced an application called TranSend which performed lossy Web image compression on the fly on behalf of users on low-bandwidth (dial-up) links, through the use of *active transformational proxies* that performed *distillation*, or datatype-specific lossy compression[1] [20]. TranSend relied on the Multipurpose Internet Mail Extensions (MIME) type [21] of an object returned from an origin server in order to determine the type of compression to be done on the inbound content. The general programming model that formed the basis for TranSend, called TACC (for transformation, aggregation, caching and customization) also supported a database of stored user profiles that allowed request processing to be customized. However, it is unclear whether these user profiles were actually used by TranSend to personalize distillation requests.

Fox et al. argued that adaptation machinery, such as the active proxies used in their approach, was better placed in the network infrastructure rather than on servers or clients, because it allowed for incremental deployment of service functionality without having to modify a very large base of servers and clients. They further stated that a centralized resource for adaptation placed within the infrastructure was more efficient in terms of cost and utilization.

[1] "Lossy compression" refers to a method of data compression that results in some (usually acceptable) level of information loss, distortion, or quality degradation in the output.

TranSend illustrated a specific example of network adaptation on flows based on capabilities of the clients (and possibly user profiles) that requested these flows. Although there was an implicit use of client capability, connectivity characteristics and user profiles as context in their model, there was no explicit architectural support for the use, processing and distribution of context in general. In addition, although a centralized resource for adaptation may be desirable from an economic point of view, the scalability, availability and reliability of such an approach is somewhat debatable.

2.2.2 Odyssey

Noble et al. introduced Odyssey, a set of operating system extensions designed to support adaptation for mobile applications running on an end-host [22]. Odyssey monitored shared resources on an end-host, particularly network bandwidth, although conceptually the design could be used to manage other resources as well. Applications expressed their resource expectations to Odyssey and were notified when these expectations could no longer be met, enabling them to adapt by allowing changes in the *fidelity*[2] of the data presented at the client.

Odyssey's approach to adaptation was characterized by Noble et al. as *application-aware adaptation*, essentially a collaborative partnership between the system and individual applications. They distinguished this from what they called *laissez-faire* approaches, where applications bear the entire responsibility for adaptation, and *application-transparent* approaches, where the entire responsibility for resource management and adaptation lied with the operating system.

Odyssey's designers claimed that application-aware adaptation provided good support for application concurrency, and enabled the management of shared resources in coordinated and cooperative way. While Odyssey promoted concurrency (i.e., the ability to execute multiple applications simultaneously) by ensuring that local resources within a single end-host were optimally shared, it did not promote concurrency among multiple end-hosts sharing a common resource, such as wireless network bandwidth, for example. In addition, Odyssey's focus on end-hosts did not provide any support for adaptation to be done by network elements – a function that would be essential in context-aware networks.

[2]"Fidelity" was defined in [22] as the degree to which the data presented to a client matched a reference copy in a server.

2.2.3 Conductor

Yarvis, Reiher and Popek presented Conductor, a system that allowed distributed adaptation within the network [23]. Software modules called *adaptors*, written in Java, were dynamically deployed along the communication path between a client and a server on Conductor-enabled nodes such as routers or on end-devices. Adaptors were specialized, implementing only a single algorithm or function, and were combined when appropriate. Each one had an input and output protocol, a set of interoperability parameters which restricted adaptor combination, and an observation component that monitored local node resources such as CPU load and link characteristics.

When a new client-server connection formed, Conductor gathered all relevant information (for some partitioning of the network) to a single node, generated a plan that was distributed to the nodes along the communication path, and finally deployed adaptors. Adaptors were typically deployed in pairs, so that an adaptor that converted the stream from protocol A to protocol B would have a counterpart downstream that converted from protocol B back to protocol A.

Conductor was *application-transparent*, that is, applications were not aware of or able to control adaptation. The rationale for application transparency, according to the designers, was to free application developers from the complexities of building support for adaptation and from the need to change or upgrade end-applications as user requirements or network technologies evolved.

Some important design approaches that may be learned from Conductor include adaptor composition, and the need to deploy adaptors (in some cases) in pairs. The need to optimally plan the deployment of adaptors, based on existing conditions within some level of network partitioning is another important concept that may be worth considering in future designs.

The Conductor design however raises some questions about the ability of the the system to quickly respond to the onset of flows, and whether the overhead of adaptor deployment can be justified in the case of short-term flows. One major assumption made by the designers was that information gathered during the planning stage was primarily connection-independent and somewhat static, and thus planning information could be cached, reducing the planning overhead. This assumption might not be entirely valid as network resources, particularly bandwidth, may change continuously, and each new connection could potentially trigger a further change in the state of these resources. A planning node would thus have no means of knowing in advance whether the observation data within the nodes had

changed significantly since the last time it had been polled (unless it applied some prediction algorithm), or know statistically if data from the nodes could be sampled without loss of information. In the worst case therefore one might expect a significant amount of traffic generated by the collection of planning information and the deployment of plans to nodes *every time* a new connection (flow) was initiated.

2.2.4 Transformer Tunnels

Sudame and Badrinath proposed the idea of Transformer Tunnels that provide route-specific adaptation functions, called *transformation functions*, operating on packet flows between two nodes of a network segment [24]. The authors claimed that packet flows that may have originally been tuned for transmission over some types of links, perhaps in terms of packet size, transmission rate, encryption method, or some other parameter, may be unsuitable for transmission over other types of links. Hence, their approach bound the adaptation function to specific segments of the network, and forced packets traversing transformer tunnels existing within those segments to undergo an adaptation appropriate to the link characteristics.

Packets entering a transformer tunnel were modified either by changing the content or by changing the way they were transmitted in order to fine tune the flow according to the segment properties. At the other end of the tunnel, packets were restored to their original form. Some examples of transformation functions tested included reassembly, buffering, encryption and compression. Reassembly involved combining small packets into larger ones in order to improve link utilization and reduce contention in shared-medium wireless links. Buffering allowed mobile hosts to put their interfaces into sleep mode and save energy, and also to recover packets lost during mobile handoff. Encryption enhanced the security of data transmitted over wireless links, and compression improved transmission performance and reduced link traffic, particularly when the link was slow enough such that the savings in transmission time were significantly higher than the compression-decompression overhead.

Although the basic idea behind Transformer Tunnels made it applicable for deployment on any suitably-configured segment within the network, the implementation presented was limited to last-hop links, typically between a mobile host and its access point or base station. A client-server architecture was used where the client (mobile host) could remotely configure tunnels at the server (the base station) in order to add or delete transformations at any time. The concept could have been more general and powerful had there been mechanisms for creating and tearing down transformer tunnels on "transformer-enabled" segments elsewhere within the network, e.g., further downstream along the flow path,

perhaps by providing a mechanism for sharing "transformation-triggering" (i.e. context) information.

2.2.5 Congestion Manager

Andersen et al. introduced Congestion Manager (CM), an operating system module that provided integrated network flow management and an application programming interface (API), that allowed applications on end-hosts to be notified of and adapt to changing network conditions [25]. CM provided integrated congestion management across *macroflows*, which the authors described as a group of flows that shared the same congestion state, control algorithms, and state information. Their rationale for operating on macroflows rather than individual flows is that multiple concurrent connections originating from single applications, such as in the Hypertext Transfer Protocol (HTTP) [26] tended to compete with, rather than learn from each other about network connections to the same receiver. These applications then ended up being unfair to other applications that used fewer connections.

CM employed a *congestion controller* that performed congestion avoidance and control on macroflows using a window-based algorithm similar to TCP's additive increase / multiplicative decrease (AIMD) scheme [27]. A *scheduler* then decided how the current window (rate) determined by the congestion controller should be apportioned among the constituent flows. User-space applications or in-kernel protocols that sent flows using CM, called *CM clients*, opened CM-enabled sockets[3], where the flow was assigned to a macroflow based on its destination. When a client requested permission to send data, CM's scheduler checked if the corresponding macroflow's window was open, and if so, granted the request and notified the client that it could proceed with transmission. When the client received feedback from its remote counterpart, it informed CM of the loss rate, number of bytes transmitted correctly, and the observed round trip time. CM then applied these parameters to its congestion management algorithm to open up the window and grant the pending request of the next scheduled flow associated with the current macroflow.

While CM's ability to share information across concurrent flows may have promoted cooperation rather than competition, this mechanism only worked within the end-host where it operated. Thus CM did not have the ability to extend the same advantages over multiple concurrent flows originating from multiple hosts. In order for that to happen, multiple hosts running CM would have had to share macroflow QoS information among each other in a cooperative fashion, within timescales appropriate to the flow lifespan – a task which

[3] A *socket* is an application programming interface that allows communication between hosts on a network. It was first developed for the Unix operating system and is widely used in systems that support TCP/IP.

would have been difficult to realize. There is a need for a mechanism (as will be presented in Section 6.2) to share macroflow QoS information obtained by CM (or a similar scheme) with network devices in order to permit the further aggregation of macroflows from different hosts, thus further transforming these competing macroflows into cooperating macroflows.

2.3 Context-aware networks

Based on the definition of context by Dey, Salber and Abowd (cf. Section 1.1, on page 15), which referred to *any* information that can be used to characterize the situation of entities [6], it may be argued that networks that used such information to trigger adaptation, such as those discussed in Section 2.2 may be considered context-aware to a certain extent. The emergence of the concepts of context-sensitive and context-aware communications, as will be discussed in Section 3.5 are marked by explicit reference to the term "context" to describe information that may trigger or influence communications. These concepts have further progressed the evolution of networks from "adaptive" to "context-aware," although the distinction is somewhat blurred. In this section, a number of architectures that may be classified as *context-aware networks* are discussed. Context-aware networks have one or more of the following characteristics:

- The use and processing of information generally considered to be or explicitly defined as context information within the network
- Systematic sensing of context information (in its broadest sense) within the network
- Provisioning of context-aware or context-triggered services within the network
- Presence of a well-defined context management infrastructure

2.3.1 The CONTEXT Project

Anagnostou et al. described the CONTEXT Project (Active Creation, Delivery and Management of Efficient Context-Aware Services; http://context.upc.es/index.htm), whose main objective was to "design, develop and assess innovative models and middleware solutions for the efficient provisioning of context-aware services making use of active network technology on top of fixed and mobile networks," enabling context-aware services to be composed and dynamically adapted for the benefit of users [28]. Anagnostou et al. listed a wide range of information types treated as context information in the CONTEXT architecture, including user information, state, preferences and agenda; location, time and date;

application information; terminal parameters; and network parameters. Jean et al. further enumerated some features of the CONTEXT architecture that allow its classification under "context-aware networks," including: (1) the integration of the context management infrastructure within the network rather than in end-applications, (2) the use of context information sensed not only from outside, but also within the network (the so-called "network context"), and (3) its focus on the provisioning and management of "network-centric context-aware services" [29], which essentially constitutes an awareness and response by the *network*, rather than by end-applications, to context information.

Although the CONTEXT architecture does not adopt a flow-centered approach to the sensing and use of context, many of its features may be of potential use by context-aware network architectures. For example, its policy-based framework for service creation, provisioning and management may be used as a means of triggering network services in response to flow context.

2.3.2 Ambient Networks and ContextWare

Karmouch [9] et al. proposed the development of an architecture called *ContextWare* and the deployment of its functional elements to manage context and mediate between context sources and sinks within networks called *ambient networks* (ANs) [30]. The role of context in ANs would be to enable automatic decision-making and adaptation within network control components and to allow externally-accessible user services (called user-facing services) to adapt their operation, thereby optimizing network service delivery, reducing complexity, and enhancing performance and functionality [31].

Two main functional entities (FEs) were defined within the ContextWare architecture, namely the Context Coordination (ConCoord) FE and the Context Management (CM) FE. The ConCoord FE was the first point of contact for any context client, such as any other functional entity within an ambient network, or external applications that required context information. A key function of the ConCoord (described in more detail in Section 6.1.1) was to provide context clients with the locations where context information may be obtained within the network.

The CM FE managed operations such as the collection and dissemination of context information within an AN. It was also responsible for scheduling and monitoring interactions between context sources and sinks, as well as allocating resources (such as communication channels) for such interactions. Other functionality managed by the CM FE concerned the processing, manipulation and aggregation of context information from its "raw" form

provided by context sources into the form required by the context client.

The ConCoord also enforced policies and resolved conflicts governing the kind and quality of context exchanged between sources and clients, and the rules governing this exchange were called Context Level Agreements (CLAs). While the notion of context clients having to request and pre-negotiate the type and *quality* of context information with the ConCoord FA seems justified within the Ambient Networks framework, it also seems to be a rather complex and possibly costly way, in terms of computational and time overhead, of obtaining context.[4] Consider for instance a mobile host (considered an ambient network in itself by definition) transiting through a third-party network, possibly for a brief period of time. Would the respective ConCoord FAs of these two networks need to go through the process of CLA negotiations for instance just to accommodate a brief flow from the mobile host? Despite these pragmatic concerns about the performance and overhead associated with context negotiation and CLAs, the idea of associating a certain quality metric with context is interesting and potentially useful, given its dynamic and occasionally imperfect nature [32].

One conceptual weakness in AN (discussed further in Section 3.1.2) was the use of the rather abstract notion of "network context" [31] without providing a clear and operationally useful definition of it. As the term was quite vague and open-ended, it could have referred to practically anything about everything within the network, thus making it difficult to separate the concerns of the ContextWare architecture from those of the other functional areas within an ambient network. For example, if the location, availability and state of services within the network were also considered forms of network context, then should the implementation of service location be placed within ContextWare? Another example might be: should the current mapping between a host name and its IP address be considered a form of network context, and if so, should name resolution be the task of ContextWare? These questions merely illustrate the need to improve or evolve ContextWare's conceptual framework, perhaps as work in the project progresses.

Despite these criticisms on some approaches taken in ambient networks, several of ContextWare's design approaches and components prove valuable in this thesis, such as in Section 6.1.1, where the ConCoord concept is used to locate sources of flow context, and in Sections 6.2 and 6.3, where functionalities similar to those provided by the CM FE – those of dissemination and aggregation – are explored.

[4] An alternative method is proposed in Section 8.3.8.

2.3.3 Situated and Autonomic Communications

Situated and Autonomic Communications (http://www.cordis.lu/ist/fet/comms.htm) is a relatively new[5] research initiative sponsored by the Commission of the European Union under the Information Society Technologies – Future and Emerging Technologies Programme. The long-term goal of this initiative is to "promote research in the area of new paradigms for communication/networking systems that can be characterised as situated (i.e. reacting locally on environment and context changes), autonomously controlled, self-organising, radically distributed, technology independent and scale-free" in order to enable the development of task- and knowledge-driven communication networks.

Sestini described two major (and perhaps overlapping) research areas that comprise this effort: self-organization in networks, and cognitive situated networking [33]. Self-organization in networks means that network elements have the capability to observe, react and reconfigure themselves to fit communication goals and environmental situations, without explicit user interaction. These elements would likely to be programmable, and autonomously controlled by policies, rules and events that lead to or facilitate certain desired group behaviors. Cognitive situated networking, on the other hand, attempts to move away from the traditional view of networks as self-contained systems of information channels, and instead recognizes that networks interact and relate in intricate ways with their larger surroundings. The use of context, meaning and semantics are seen as key approaches in cognitive situated networking, bridging the gap between the physical and digital worlds, and ensuring that communication technology takes full advantage of the setting and environment in which it is being used [33].

The emergence of this research effort highlights the relevance of the work described in this thesis within the larger research context. A section in [33] states: "The final step will be leveraging meaning into the network, using semantics and context ... For instance, the notion of adding semantic tags to information exchanges – letting the network "know" what it is transporting – can be a key enabler for adaptive and intelligent behaviour." Flow context is envisioned to provide exactly this function, and perhaps even more, as will be seen in Section 3.3, where flow context is defined to encompass information not only about the flow itself (i.e., "what it is transporting") but likewise information about other entities that are also relevant to or associated with the flow, such as the users that generate the flows, their activities, their devices, the applications, and others.

[5]Projects funded under this initiative commenced in January 2006.

2.4 Chapter summary

This chapter presented a critical review of relevant prior and ongoing work, including the architecture of context-aware applications, adaptive networks, and the recent work in context-aware networks. The features of these architectures were summarized, and when possible, related with the relevant aspects of this thesis.

It had been mentioned that adaptive networks served as precursors to current context-aware networks. The next chapter re-examines some of the lessons learned in the review presented in this chapter, and identifies the particular context-handling functionalities and architectural components needed in order to evolve yesterday's adaptive networks into tomorrow's context-aware ones.

Chapter 3

Understanding Flow Context: Concepts

3.1 Context

Context is defined in the New Oxford Dictionary of English as "the circumstances that form the setting for an event ... and in terms of which it can be fully understood and assessed." Within the related domains of human-computer interaction (HCI) and pervasive and ubiquitous computing, the term context has been used to refer to a number of things, including: the location and identities of nearby people and objects relevant to an application [34]; the elements of the user's environment that a computer may know about [35]; or knowledge about the state, situation and surroundings of the user and her devices [36, 37]. A well-accepted definition of context is given by Dey, Salber and Abowd as "any information that can be used to characterize the situation of entities ... that are considered relevant to the interaction between a user and an application, including the user and the applications themselves." [6].

In context-aware computing, applications (called *context-aware applications*) discover and take advantage of context information to present information and services to a user, execute services automatically, or provide information tags for later retrieval and use, based on the relevance of this information or services to the user's task [6].

3.1.1 Entity-centered context

One common characteristic of context-aware applications in the HCI and pervasive computing domains is that context information is centered around or bound to particular *entities* of interest. As shown by the synonyms and definitions of context given in the previous section, these entities of interest are usually the user or the application. It is relatively easily to conceptualize, say, the context of a user, because a user is essentially a single entity that can easily be distinguished from its surroundings or environment, and although humans have complex behaviors, it is still possible to come up with concise descriptions of their situation, disposition, social setting, or activity, i.e., context. Thus, one frequently encounters the following contextual description of humans in pervasive computing applications such as "user A is in a meeting," or "user B is in location X."

Similarly, Schmidt argued for the use of entity-based contexts rather than system-based contexts:

> "The domain knowledge about a specific entity is more universal and easier to establish than the domain knowledge of a complex system, and hence it is simpler to identify and implement contexts on entity level than on system level." [11]

This however does not imply that system-level context should not be inferred from more elementary entity-level contexts. On the contrary, one may use a *bottom-up* approach to sensing context, where information and domain knowledge would progressively be built up from simpler and more manageable pieces of information coming from sensors working at the level of entities [11]. If one were to apply this design principle to the use of context within networks, it would mean that one should first identify specific entities of interest within the network, sense their context, and attempt to infer higher-level system contexts from these.

3.1.2 The notion of "network context"

Recently there have been some efforts to introduce the notions of "network-related context" as well as "network-centric context" [7, 9, 31, 38]. One possible distinction between this type of context and the traditional type of context considered in context-aware computing is in terms of *where* the context is used to trigger changes: network-related context triggers changes primarily within the network rather than in host applications [38]. Another distinction revolves around the class of entities whose operational state are of interest:

> "The relevant entities of network-related context information include network nodes and network parameters. Examples of context information include the location of a node in the network topology, its own resources, network resources available to it, and its other states (for example, mobility, security, etc.)." [9]

Yang used the term "network-centric context" to refer to context information where the entities of interest may be physical devices such as routers or switches, or virtual objects such as network services [7]. His view of context is "any information, obtained either explicitly or implicitly, that can be used to characterize one certain aspect of an entity that is involved in a specific application or network service."

It is difficult to see from these definitions how network-related or network-centric context are different from, say, information gathered and processed by contemporary network management systems such as those using the Simple Network Management Protocol (SNMP) [39]. While some definitions included the notion of inferred information about the state of network-based entities [7], it is unclear if this is what differentiated network-centric context from any other information currently obtained and processed by contemporary network management systems. Is SNMP data different from network context? Or is it merely a subset of what is now considered network context? Does it have to be processed before it can be considered context?

It is also somewhat difficult to apply the entity-based definition of context by Dey, Salber and Abowd (cf. Section 1.1, on page 15) using the *network* as the entity of interest in order to arrive at a notion of "network context." One reason is that a network is composed of multiple sub-entities such as nodes, links, and end-hosts; one may as well include all of the traffic flowing among the physical devices or the services provided by the network as entities of interest if one pertains to networks that are actually operational. Thus when one says "network context" it is rather difficult to identify the specific entity of interest, or to isolate what is considered to be its environment or situation, or to arrive at an aggregate description of the situation – context – of the complex entity collectively referred to as the network. As an example, when users sometimes complain that "the network is congested," they may actually be making an observation based on high error rates on a specific transmission link, packet loss within a single router, or congestion along a particular path in the network. Another user whose application flow might be traversing a different path may not encounter the same conditions, and therefore arrive at a different conclusion. Thus, some contextual descriptions or observations about the network may not necessarily be true or applicable on a global, system-wide sense.

Context type	Definition
Entity-level context	Context information centered around a particular entity. For example, the location of a single (human) user.
System-level context	Context information that has been collected and processed to describe an entire system composed of multiple entities. For example, the collective state of the network from a user's point of view ("the network is congested").
Network-related context, network-centric context	Context information that describes or pertains to entities within the network. For example, the state of nodes within the network, or their location in the topology, or their available resources.
User context	Context information characterizing a single user.

Table 3.1: Some views for context.

Finally, there is a danger that the terms "network-related context" or "network-centric context" may tend to exclude, albeit unintentionally, the more traditional forms of context such as user context. Ultimately, users are the ones whose applications generate traffic over networks, so it is logical to assume that their activities – as described explicitly or implicitly by their contexts – affect network state and impact network performance. For example, a user accessing the Internet from home for recreational purposes may exhibit a different network traffic profile than the same user working at the office, or when traveling between these two locations.

Table 3.1 summarizes the different views for context so far discussed in the preceding sections.

3.2 Flows as entities of interest

In Chapter 1 it was stated that it was the vision of this thesis to enable minimally-distracting networks to be built. A user however may be distracted as a result of explicit interaction with the network. Thus, to minimize this distraction, this user-network interaction needs to be examined more closely.

When a user runs a networked application on an end-host, the application generates a sequence of packets, traditionally called a flow [41]. The flow traverses network links and nodes, where it may be switched, buffered, routed, queued or processed in some other way. In addition the flow may suffer packet loss, delay, or its constituent packets may be subjected to variable delays resulting in what is called jitter.

The flow is the physical (or rather, electronic) embodiment of the user's interaction with the network. Furthermore, in many cases, the conditions experienced by the flow directly

affect the user's experience, despite the attempts of most architectures to abstract or hide the details of low-level interactions through mechanisms such as retransmission (such as in the Transmission Control Protocol, or TCP) and error correction. Some examples of network conditions producing perceptible effects on users include network congestion producing large delays in downloading files or Web pages, insufficient bandwidth or packet loss resulting in the degradation of real-time audio or video streams, or large transmission latencies resulting in annoying echoes in two-way voice over IP (VoIP).

Equally significant is the fact that flows are distinguishable and identifiable entities, so it is also possible to view and analyze them separately from their environment, consequently making it possible to identify the elements that constitute or describe their environment or context.

3.2.1 Defining "flows"

Multiple definitions of the term *flow* and its variants exist, usually shaped by the domain where the concept is applied. Clark originally conceptualized it as an alternative building block for the Internet, defining it broadly as a sequence of packets traveling from a source to a destination [40]. This sequence of packets may typically be the result of the activity of a single entity [41].

Subsequent definitions that emerged from the network measurement community focused on defining the set of attributes that allowed packets to be classified into flows for measurement purposes. A well-known definition for instance uses the 5-tuple ⟨IP protocol, source IP address, source port, destination IP address, destination port⟩, delimited by a fixed timeout interval [42]. Brownlee considered flows to include arbitrary groupings of packets defined by the attributes of endpoints that in turn may be described by a complete 5-tuple, a pair of netblocks, or two lists of netblocks [43]. A more general, parameterized definition by Claffy, Braun and Polyzos further took into account the directionality, endpoint aggregation, endpoint granularity, and protocol layer of the packet sequence [42]. Flows may be further qualitatively characterized by their size (mice and elephants) or their lifetime (dragonflies and tortoises) [44]. Likewise associated with the notion of flows are *streams*, made of bi-directional 5-tuple flows, and torrents, comprising all traffic on a link [43]. In addition, the term *session* could refer to a data flow defined by the triple ⟨DestAddress, ProtocolID, [, DstPort]⟩ [45], or a data flow delimited by transport-layer protocol semantics (e.g. SYN and FIN for TCP) or end-application initiation and termination.

In the EU-IST Ambient Networks Project, the terms flow, bearer and session referred

to connectivity abstractions [46]. A flow was defined as a unidirectional data exchange between endpoints that were identified by a pair of locators, and was usually constrained to a single network technology. A bearer, on the other hand, had a more end-to-end character, and its endpoints were not bound to locators but to higher-level objects in the AN naming framework. Sessions, which were associated with end-applications, combined multiple bearers to form custom transport entities for use by the application.

Given the wide range of operational and application-dependent definitions of the term "flow" and its variants, and the fact that flows, in reality, do exist at various levels of abstraction (e.g. at different "layers," such as network-layer flows, transport-layer flows, application-layer flows, and others), a broad definition of the concept is adopted instead and presented below:

Definition 1. *A **flow** is a distinguishable and related sequence of protocol data units (PDUs) transmitted across a network.*

In Chapter 4, this broad definition is further divided into specialized subclasses through a formal taxonomy of flows, classified along a range of parameters called flow context. Clearly however, the scope of the "flow" concept used in this thesis involves only PDUs transmitted across a network: for example, at the link layer, it may involve a flow of frames; at the network layer, it may involve a flow of packets; at the transport layer, it may include a flow of datagrams or segments; at the application layer it may involve an exchange of application messages.

3.3 Flow context

3.3.1 Motivation

Given the definition of flows in the previous section, what sort of information might then constitute flow context? As examples, the following information about a flow or group of flows might be of interest in a network that adapts to context in order to optimize user-network interactions:

- What kind of links did the flow encounter within the network? Were they high-speed or low-speed links? Were they congested? Were they relatively error-free, or do they have high error rates? Were they secure links?
- Did the flow encounter significant switching or queueing delays within the network

nodes or devices it traversed? Were the router buffers full, resulting in packets being dropped?

- What are the traffic characteristics of the flow? Does it have a "bursty" (highly variable bit rate) traffic profile, or does it have a fairly constant bit rate? Does it tend to have a relatively constant bit rate, regardless of the bandwidth or transmission rate of the link, such as in some classes of streaming audio or video, or does it seek to occupy most of the available bandwidth, such as in TCP flows?

- What kind of application generated the flow? Was it a real-time, interactive voice application? Does it require a certain amount of guaranteed bandwidth or bounds on latency? Can it be given best-effort treatment within the network? Can the flow withstand a certain level of packet loss? Can its average bit rate be reduced somehow? Is it a malicious flow, or of suspicious or dubious nature?

- What were the activities of the user who generated the flow? Where is she located? Is she moving? If she is moving, will the flow have to be rerouted? Will it originate from or terminate at a new access point within the network?

- What kind of device generated the flow? Does the device have limited amounts of power, such that it cannot afford to retransmit the flow?

These questions help illustrate the definition and specific instances (examples) of flow context, and highlight the motivation for using it. Each case describes information that helps characterize not only the flow itself, but also the factors that affect the flow's properties or state.

The main motivation for developing a broad concept of flow context is that it is advantageous to be able to view flows within the context of their environment. Since the flow is the embodiment of the interaction between the user and the network (cf. Section 3.2, page 37), by extension, flow context also serves to characterize this user-network interaction. By using this information, a context-aware network can potentially adapt or respond properly, and optimize the interaction between the user (or group of users) and itself.

The intrinsic properties of flows, particularly parameters that describe its traffic characteristics, are already extensively used in network management and traffic engineering [42, 43, 44, 47, 48, 49]. However, context-aware networks may also benefit from information describing the state, characteristics or properties of external entities that affect or may have influence over the flow in order to better characterize or "understand" the nature of the flow, or predict its behavior, for adaptation or optimization purposes. For example, the

knowledge that a flow is being routed over a congested link may help explain why its loss rate is high, or why its bandwidth is low. Equivalently, the knowledge that a set of flows are generated by certain application classes, such as peer-to-peer or malicious applications, may help networks predict their traffic volume and resource impact, allowing solutions for mitigation [50] or protection [51] to be deployed or activated. This rather broad conceptualization and scope for flow context differentiates it from, say, the limited mapping of flows to QoS-related parameters typical of the early work in network adaptation [18, 19]. Because such early work was limited to mapping flows to their QoS characteristics and negotiated service contracts, the range of adaptations within the network was necessarily limited to QoS-related actions. Network adaptation dealing with, for example, user mobility, the need for security and privacy, were generally not supported or investigated in such early QoS-centric work.

Furthermore, if flows can be explicitly associated with the relevant external entities within their environment, then context-aware networks may attempt to infer information about the state, properties, behavior or intention of these external entities through an examination of the flow, and use this information in order to improve user-network interaction, properly manage resources, adapt or optimize operations, or to protect themselves. For example, the type of applications generating the flow may be determined through an examination of the flow itself [52, 53], or the implicit intention of the user inferred from it, such as in the cases of email filtering for spam [54] and network intrusion detection [55, 56].

It is sometimes surprising how much information about entities that are external, or *extrinsic* to the flow, but which are related to it, can be inferred through an examination of the flow. For example, Kumar, Paxson and Weaver, through an analysis of limited and imperfect flow data attributed to the Witty worm, coupled with knowledge about the algorithmic behavior of the worm's code, were able to deduce surprisingly detailed and accurate information about infected hosts, such as "who infected whom," which network was specifically targeted (despite an effort to obfuscate this through a world-wide, randomized infection pattern), the initial point of infection, each host's access bandwidth, the last time they were booted up, and even the number of disks attached to infected hosts [57].

The primary objective therefore of defining and developing the concept of flow context is to provide the means by which flows can be viewed and characterized both in terms of their internal properties, called *intrinsic context*, and their external environment, called *extrinsic context*. This would be achieved by (1) explicitly associating flows with their contexts, both intrinsic and extrinsic, and (2) by providing the means by which this broad notion of flow context can be modeled and represented. The vision is to enable the creation of context-aware networks where the statements "flow A has bandwidth X" and "flow A is

destined for user Y, who is watching video" can be represented and used.

3.3.2 Definition

Recalling the entity-based definition of context put forth by Dey, Salber and Abowd (cf. Section 3.1, on page 15), the context of a network entity called a flow is defined as follows:

Definition 2. ***Flow context*** *is any information relevant to an interested entity, that can be used to characterize the situation of a flow. It includes information pertaining to other entities and circumstances that give rise to or accompany the flow's generation at the source, affect its transmission and processing through the network, and influence its use at its destination.*

This definition includes not only the intrinsic, low-level characteristics of a flow, but also the nature of the applications, devices, and the activities, intentions, preferences and identities of the users that produce or consume the flow. Flow context may be directly sensed, inferred, processed or aggregated into other forms of flow context.

Note that the above definition also involves the notion of an *interested entity.* The entity may be a user or group of users, a class of end-applications, a network device, a user device, network middleware, the network operator, or some other entity that may have some use for the context information. The purpose of introducing such an entity, although it may be abstract (rather than a specific and identified individual) is to limit the scope and purpose of the context being collected. While the notion of context information is purposely very broad by definition [10, 6], its scope is limited in practice by its *relevance* to its intended consumer (i.e., the interested entity). This degree of relevance is described, along with its other characteristics such as its fidelity, precision, frequency, timeliness, and other quality attributes, by a concept called quality of context (QoC). QoC is an important parameter that is further discussed in the next section of this thesis.

The conceptualization of flow context is expected to become more clear as the discussion in this thesis progresses, particularly in the next section, where its characteristics are discussed; in Chapter 4, where the concept is formally modeled; and in Chapter 5, where some specific examples of flow context are presented and mechanisms for sensing them are described. On the other hand, its usefulness will be demonstrated in Chapter 7, where its application in areas such as implicit QoS signaling, intelligent flow classification and management, mobility and moving networks, overlay routing, content delivery, attack mitigation and control, network security, privacy, accounting, billing, and other areas of network management, are described.

3.3.3 Characteristics of flow context

In this section some of the characteristics of flow context are examined. Since the definition of flow context also includes information defined as "context" in the domain of context-aware *applications* – such as user activity or location, or the capabilities of user devices – flow context may therefore share many of the characteristics of these types of context. However, there are other characteristics unique to flow context, which are examined as well.

Henricksen, Indulska and Rakotonirainy suggested that context has the following characteristics: (1) it can either be dynamic or static, (2) it tends to be imperfect, (3) it has many alternative representations, and (4) it is highly interrelated [32]. These characteristics as they apply to flow context are examined below:

Flow context may be dynamic or static. Flow context is conceptually more dynamic than static, because flows – even those that exist for relatively long periods of time – exist only on a temporary basis within the network. In mobile ad-hoc networks, the mobility of participating hosts alone can continuously alter the path of a flow, and thus by definition, alter its context. The internal dynamics of flows and their cross-interactions may also result in highly variable behavior. Network traffic levels, for example, are highly variable and "bursty" even at various levels of aggregation [58, 59, 60]. However, within the duration that a flow exists, *some* pieces of information may remain static: for example, if a flow traverses an invariant path within a fixed network, certain properties of the node, such as the type of the interfaces encountered along that fixed path, would remain the same. Thus, while some aspects of a flow's context may be dynamic, others might be considered relatively static.

Flow context is bound to be imperfect. The accuracy, precision, consistency and completeness of flow context depends on the ability of a system to reflect the true state of a flow or macroflow, and that of its environment. Although the individual bits and the protocol data units (such as frames, cells or packets) that constitute flows are arguably discrete and countable, physical limitations in the monitoring and measurement instrumentation may lead to inaccuracies in sensing and interpreting flow context. Monitoring large numbers of flows can be a slow, inaccurate and resource-intensive activity [61], often forcing the use of statistical sampling (i.e., inferring the behavior of the whole from an analysis of a subset) or the use of algorithms that favor speed and efficient execution possibly at the expense of accuracy [62, 63]. In some cases, such as in the area of flow

classification (i.e., classifying flows into groups of flows that share similar features, such as similar types of media payloads or application content), statistical techniques yield results that are typically no more than 50-70% accurate, although approaches that achieve up to 95% accuracy have been reported [64].

In addition, flow context also includes information about entities that may be external to the flow itself, such as the activities or intentions of users. It is often difficult to accurately deduce the high-level activities, much less the intentions, of users who generate the flow simply from an examination of the flow itself. However, this is often the implicit high-level objective of some systems, especially those that deal with network security. For example, intrusion detection systems such as Snort attempt to identify potentially "hostile" or "malicious" network flows by examining the contents and characteristics of the flow itself [55], and in doing so, implicitly attempt to infer the intentions of the users generating them.

In relation to the imperfect nature of flow context, some authors used the important notion of *quality of context* (QoC) to model some objective and subjective quality attributes of context information, such as its fidelity, precision, accuracy, trustworthiness, resolution, frequency, and timeliness [10, 31, 32, 65, 66, 67, 68, 69, 70, 71].

Flow context can have multiple representations. A flow's context may be described or represented in a variety of ways. For example, the rate of network traffic generated by a flow might be represented in either kilobits per second or megabits per second, reflecting a simple (but important) distinction in measurement units. However, network traffic may also be validly expressed in terms of packets per second (for packet-based networks), which, due to variable packet lengths, might not map in a predefined way to a traffic rate representation in bits per second – to convert the former to the latter may also require some knowledge of the packet size distribution within the flow. In addition, there may be variations in contextual representations due to the level of abstraction in use. Using flow rate once again as an example, qualitative descriptions such as "large flows" or "low-bandwidth flows" may conceivably be more useful or desirable in some situations, rather than a numerical representation.

Flow context is composed of interrelated information. A wide variety of factors and entities influence the generation, transmission and consumption of flows, and these often interact with or are interrelated with each other. For example, a user who wishes to watch a certain movie might attempt to do so by opening a video application and downloading a video stream, possibly generating a significant level of traffic over the network. If the network is congested, resulting in poor video quality, the user might decide

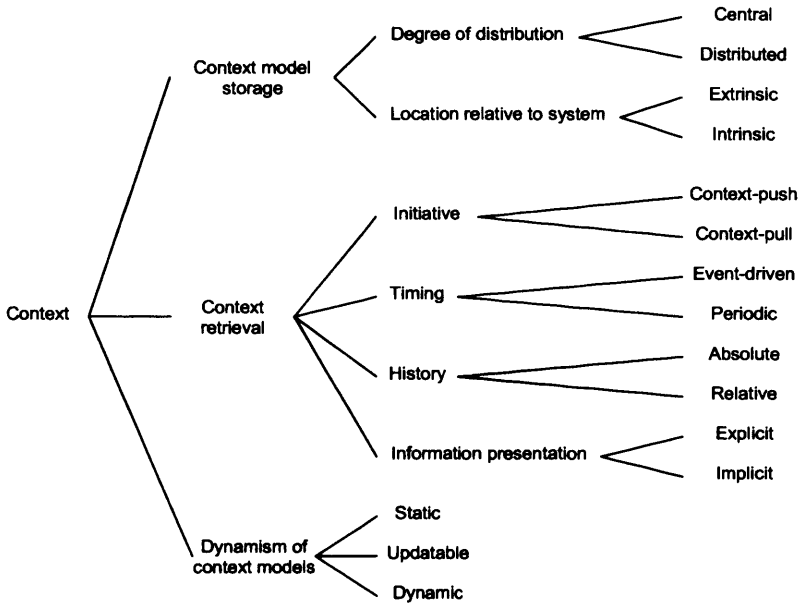


Figure 3.1: Context classification framework by Siljee et al. [72].

to terminate the stream. This example illustrates the relationships and possibly complex interactions between different forms of flow context, such as user intention and activity, existing network conditions, flow traffic levels and application-level quality of service.

Identifying the characteristics of flow context is useful in designing systems that are intended to handle it. Siljee, Bosloper and Nijhuis [72] presented a classification framework intended for systems that store and retrieve context. Their framework, shown in Figure 3.1 focused on three main areas, namely, the manner in which the context model is stored, how context is retrieved, and the dynamism of the context model.[1] Although many of the characteristics identified in this framework may also be found in the work of Henricksen et al. discussed previously, the following additional set of characteristics are implicit in the framework by Siljee et al.:

Flow context may be intrinsic or extrinsic to the flow itself. The basic definition of flow context includes both *intrinsic flow context*, which pertains to information about the flow's internal state, and which is usually obtained directly from an examination of

[1]Their use of the term "context model" refers to the entity or process that translates raw data from context sensors into useable context information.

the flow; and *extrinsic flow context*, which deals with information about entities that are external to the flow, but which nonetheless influence the flow's state.

Because of interactions and interrelationships between different classes of flow context as discussed in the previous sections, it is sometimes difficult to classify information strictly and exclusively into one of these two types. For example, one can attempt to infer user activity, which may be classified as extrinsic context, from a flow's content, which by definition is intrinsic context, as is the case in network intrusion detection systems such as Snort [55]. In addition, it should be noted that the classification into intrinsic and extrinsic context is different from [72], where the terms are used to denote whether a system *stores* context information internally or externally, respectively.

Flow context may be implicit or explicit. Flow context information may either be *explicit*, that is, the information directly sensed and provided by an appropriate piece of hardware or software, called a sensor, is already in a form that may be directly used. On the other hand, when context information needs to be processed, aggregated, or further inferred, possibly in conjunction with other pieces of information, this may be called *implicit* context. Such aggregation may include numerical, logical, or some other form of algorithmic processing, validation, range-checking, conversion from one format to another, or inference and reasoning. Aggregation may be performed over time (temporal), across flows, or with other pieces of information. Raw data may also be used to infer the occurrence of *events*, and the confluence of multiple events may lead one to infer more complex events, or *situations*.

Figure 3.2 illustrates how different pieces of context may need to be aggregated before they can be used by a potential consumer. In this example, raw data on user activities, packet flows and network links are sensed from user and network devices. The information gathered from the appropriate sensors is aggregated, allowing a system to infer user-flow mappings, per-flow QoS data, per-flow content profiles, and link capacities. The next level of processing correlates the flow content profiles with the set of user privileges, and determines the relative importance of the user's activity and traffic on the network. On the same level the traffic data from other flows is aggregated to yield a picture of the total amount of traffic traversing the link. It might be the case, as in this example, that the total amount of traffic, when compared to the link capacity (inferred from the interface type), indicates a situation where the link is approaching congestion. The context consumer in this case now has the pieces of information that are relevant to its decision-making, namely, the fact that (a) the link is congested, (b) $flow_x$ is consuming a considerable amount of bandwidth, and (c) $flow_x$ has low priority. The context consumer would then presumably

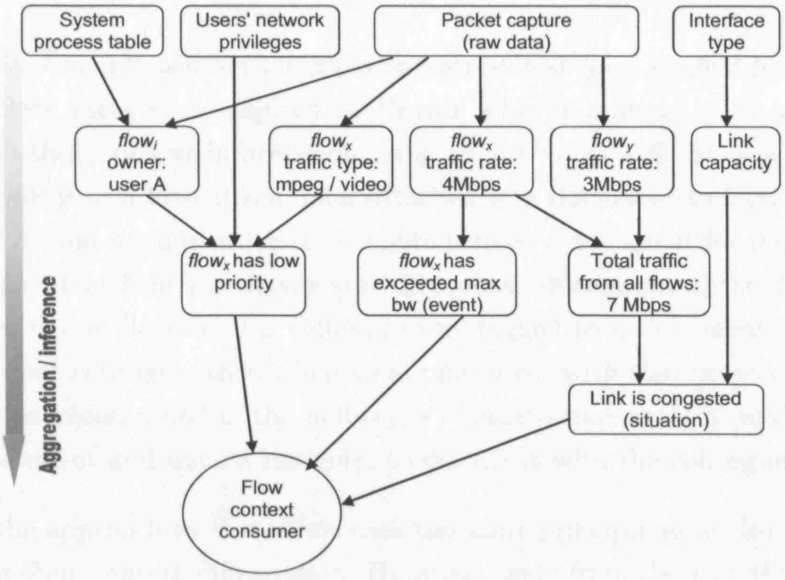


Figure 3.2: An example of context aggregation.

initiate some actions that would address such a situation.

It should be noted that many of the axes of classification or characterization previously mentioned do not provide strict dichotomies, as in "dynamic vs. static" or "intrinsic vs. extrinsic" or "explicit vs. implicit." It might be useful instead to envisage varying degrees relative to some observer or user of the information. Using Figure 3.2 again as an example, the information that $flow_x$ may be transporting an MPEG video stream might be sufficient (i.e. complete) from the point of view of a network entity whose task is simply to perform video stream adaptation. However, that same piece of information would be considered insufficient (incomplete) by another entity whose task is prioritize flows based on both their content types and the privileges of their respective users.

The foregoing discussion on the characteristics of context as enumerated by other researchers in the field, and the subsequent attempt to examine these characteristics with respect to flow context, hopefully leads to a better understanding of flow context. In addition, it is also a critical prerequisite to the development of appropriate *models* for flow context, presented in Chapter 4.

3.3.4 Context-tagged flows

In this section two complementary concepts are introduced: *flow context tags*, and *context-tagged flows*. Both ideas were inspired by Brown, who presented the idea of associating documents with their context information using *stick-e notes* [73]. Stick-e notes were the electronic equivalent of a Post-it reminder attached to a document to facilitate the future use or retrieval of that document. A stick-e note primed a document for triggering when a current context matched the previously stored context information in the document's tag. As an example, if the identity of a colleague was tagged to a document that had to be discussed with that colleague, then when an appointment with that person was scheduled, or the person was encountered in the hallway, a context-aware system would immediately retrieve the document and issue a reminder to discuss it with the colleague.

Conceptually the approach in this thesis uses the same principle as stick-e notes: *entities are tagged with their context information.* However, aside from the fact that stick-e notes and flow context tagging operate in different application domains, the nature of the entities that are associated with stick-e notes and flow context tags are vastly different: documents (with which stick-e notes are associated) are persistent and typically long-lived entities, while flows (with which flow context tags are associated) are ephemeral and may be short-lived. Thus, while the primary application of context tagging in stick-e notes is for *future* retrieval of a document, context tags for flows are expected to be used while the flow exists.

The early work expressing the idea of "flow context tagging" that appeared in [74], and subsequently in [75, 76, 77], were based on the foregoing discussion on Brown's stick-e notes. However, these also contained a specific model on how flow context tags were to be disseminated in the network (to be discussed further in Section 6.2.2). As the work in this thesis evolved, it was realized that other context dissemination models may apply, the concepts were subsequently modified to account for the wider perspective. Consequently, the following definitions apply in this thesis:

Definition 3. *A **flow context tag** is an explicit representation or encoding of a flow's context and its association with a particular flow instance.*

Definition 4. *A **context-tagged flow** is a flow whose context has been sensed and explicitly encoded using a context tag.*

Flow context tags realize the earlier-stated objectives (cf. Section 3.3.1) of encoding and representing both intrinsic and extrinsic flow context, and explicitly associating flow instances with their context. Since flows occur at different levels of abstraction and aggregation, that is, a "flow" could conceptually refer to a *macroflow* (i.e. a bundle of flows), it

is possible for a single tag to encode the common characteristics of the flow bundle. For example, a set of flows destined for a set of different users, each containing (say) the same video stream, could be considered a macroflow, and a tag may be defined, describing the common attributes (e.g. video payload characteristics such as the encoding type) of the entire macroflow.

3.4 The flow context life cycle

This section describes the various stages in a notional life cycle for flow context, from the moment it is obtained from information sources, until it is in a form that can be used and accessed by context consumers. To develop this life cycle, the adaptive and context-aware architectures reviewed in Chapter 2 are revisited, and some of the discussions and lessons learned from the study of these architectures are restated and summarized. This is followed by a brief description of the life cycle itself, with references to more detailed discussions of each of the stages elsewhere in this thesis.

3.4.1 Lessons from adaptive and context-aware architectures

Many of the early adaptive architectures were designed only to handle "simple" forms of context information, that is, they did not explicitly address the particular nature and requirements of handling more complex forms of context. From the review of context-aware architectures in Section 2.1 it was shown that oftentimes, several intermediate steps in a layered architecture are needed in order to process raw data from *sensors* into useful context. Some of the processing steps needed after raw data acquisition include augmenting new sensor data with existing information [14], algorithmic computation and normalization (i.e., transformation into a standardized representation) [15], fusion or aggregation of relevant multi-sensor data and processed context [6, 16], interpretation of low-level data into higher-level information [6], or abstraction of low-level data into symbolic cues and contextual situations [11]. While some have used algorithmic techniques to achieve these steps [15], others propose the use of artificial intelligence and machine learning techniques in order to process raw context information into "higher-level" useful knowledge [78, 79].

Another useful lesson learned from prior work is the need to carefully plan the deployment of services, such as adaptation services. Adaptation machinery may be placed within the network to allow service functionality to be incrementally deployed with minimal modification to existing end-hosts [23], or to perform link- or route-specific adaptation [24].

However, since adaptation should be a collaborative partnership between the system (operating system and networks) and end-applications [22], a mechanism is needed by which context information, used as basis for adaptation, may be shared between end-hosts and the network. Such a mechanism has either been absent or not yet fully-developed in these early architectures [20, 22, 25].

In some cases, service modules, particularly adaptor modules, may have to be dynamically composed and sometimes deployed in complementary pairs in order to support data transparency [23]. For example, if an encryption service module is deployed at one end of an secure segment, a corresponding decryption module should be installed at the other end in order to restore the information to its original format. A policy-based framework may be one approach for expressing service deployment plans, while active networking may be considered for dynamic and flexible service deployment [7, 23, 29].

3.4.2 A notional life cycle for flow context

From the foregoing discussion, it is apparent that the first step in obtaining useful context information involves *sensing*, or the use of context sensors, defined in this thesis as any software or hardware that can perform context sensing. The issue of sensing flow context is discussed in more detail throughout Chapter 5.

If flow context is primarily obtained from context sensors, then entities that either manipulate, process or consume this information should be able to discover or locate these sensors first. These steps are called *discovery* or *location*, and are discussed in Section 6.1.

To be able to transfer context information among the different entities that sense, manipulate, process, and consume, a scheme for flow context *dissemination* should be in place. The issue of dissemination is tackled further in Section 6.2.

Raw context information obtained by sensors and disseminated through the network may have to be further processed or aggregated to transform it into a form that is useful or relevant to the consumers of the information. The different types of processing described in the previous section are collectively lumped under the terms *aggregation* or *processing*, and discussed further in Section 6.3.

Finally, flow context may be used by context-aware elements within the network for various purposes. The *use* and applications of flow context are explored in Chapter 7.

Note that these stages do not necessarily occur in this order, and may be performed iteratively, recursively, in combination, or in a modified sequence. For example, a context sensor

may also inherently perform some internal processing or aggregation before it disseminates its information. However, it is logical to expect that the sensing of raw or low-level information would be one of the first steps in most systems. On the other hand, an information aggregator might be viewed by other consumers of context information as a source or a sensor, so it is also possible have sensing to exist at other levels of the life cycle as well.

In Section 3.3.4, the concept of *flow context tagging* was defined as the association of a flow with an explicit representation or encoding of its context. In order to arrive at such a representation or encoding, the flow's context first has to be sensed, and in some cases, disseminated and then aggregated or further processed. Flow context sensing and aggregation, and therefore tagging, as will be explained in Chapters 5 and 6, can be performed either at the network's edge or within the network itself. In addition, since the flows involved may exist at different abstraction layers, the context sensors may also exist at these different layers as well. For example, for network- (i.e., packet-) level flow context sensing, the sensing of the relevant flow context might occur at the network layer. Conversely, for application-level flows, the appropriate sensors might exist within the applications themselves. However, it is also possible – and envisioned by this thesis – for flow context sensing to be done in a multilayer and multi-entity fashion. Again, this is discussed in more detail in Chapter 5.

3.5 Related concepts

This section discusses some concepts that are either similar or related to flow context, flow context tagging, or context-tagged flows.

3.5.1 Context-aware communication

Henricksen, Indulska and Rakotonirainy used the term *context-aware communication* to refer to the use of context in communications applications [32]. This definition was quite vague, so the applications cited as examples by Henricksen et al., which included the *Context/Communication Information Agent (CIA)* by Hong and Landay [80], the *Context-Call* application by Schmidt, Takaluoma and Mäntyjärvi [81], as well as four other applications built under the theme *Everywhere Messaging* by Schmandt et al. [82], were examined in detail.

CIA [80] was an autonomous software agent that helped find and deliver "the right information at the right time" through the use of context information. An early prototype

built by Hong and Landay took speech input, processed it through a speech recognizer, and performed Web searches based on keywords spotted in the recognized speech.

In Context-Call [81], an application provided a user, prior to making a telephone call, the ability to view the context (social situation) of another person (a potential call recipient) using the Wireless Application Protocol (WAP). Based on the indicated context of the potential call recipient (Free, Meeting, Working, at Home, or BUSY!), a potential caller could then decide whether to place the call, leave a message, or cancel the call. A similar feature may also be found in popular instant messaging (IM) applications such as Yahoo! Messenger (http://messenger.yahoo.com/), where similar status messages such as Available, Busy, Stepped Out, On the Phone, and other user-defined messages provide potential message senders context information about the intended recipient. More sophisticated contextual cues were provided in *ConChat* [83, 84], developed by Ranganathan et al., where users were provided information on the activities, situation and surroundings of other parties in an electronic chat conversation, using first-order predicate calculus and Boolean algebra to internally represent and process context obtained from a "smart space" infrastructure called *Gaia* [85].

Everywhere Messaging [82] envisaged the ability to engage in electronic messaging in a way that was unintrusive and adaptive to the social situation and location of the receiver. Four projects were described under this theme. *Clues* was an email filtering agent that analyzed information on a user's desktop computer in order to identify "timely" messages. *Active Manager* used Clues to prioritize incoming email and forward each message to an appropriate communication channel such as a pager, fax machine or phone based on the user's preferences and situation such as location and time-of-day. *Nomadic Radio* analyzed the user's attentive state and determined the extent to which a user may be interrupted by auditory messages such as voice email, email, news and calendar broadcasts. *comMotion* tracked user location using the Global Positioning System (GPS) and learned travel patterns in order to provide "right time right place" messaging.

In the examples cited, the word "communication" in the phrase *context-aware communication* pertained to high-level communication between humans, either at the application layer, or even direct person-to-person conversations between users. In addition, the primary consumers of context information were end-applications, rather than network devices.

3.5.2 Context-sensitive communication

Yau et al. used the term *context-sensitive communication* (CSC) to refer to context-triggered impromptu and possibly short-lived interactions between applications in ubiquitous computing environments [86, 87], or spontaneously-established communication channels for this purpose [88]. The term was used in describing the operation of a type of middleware called *Reconfigurable Context-Sensitive Middleware* that allowed applications to dynamically discover and interact with other applications that may possess required context information. Khedr and Karmouch extended the concept in order to define a *context-sensitive communication protocol* (CSCP), which they described as a "communication protocol sensitive to the frequent changes in the environment ... capable of creating, modifying, adapting sessions and persevering the context-based preferences of the sessions" [89]. CSCP was basically an extension of the Session Initiation Protocol (SIP) [90] supporting the context-based creation and management of sessions between entities and the dissemination of context information between them.

In the Ambient Networks (AN) project (discussed in Sect. 2.3.2) context-sensitive communications were defined as a type of communication where channels were established between devices based on some specific contexts [9, 38]. These were envisioned to provide the means by which communication protocols were kept continuously aware of the current situation in their environments [31]. Unlike in the previous examples where context-aware or context-sensitive communications were used to disseminate context primarily for use by applications [32, 80, 81, 82, 83, 84, 86, 87, 88, 89], context-sensitive communications were used in ANs for context dissemination to network entities.

3.5.3 Situated and autonomic communications

In Section 2.3.3 the emerging concept of situated and autonomic communications was briefly reviewed. As previously mentioned, this paradigm refers to communication and networking systems that react locally on environment and context changes, are autonomous, self-organising, distributed, technology independent and scale-free.

3.5.4 Positioning context-tagged flows

Although the terms "context-aware communications" and "context-sensitive communications" seem to be synonymous (and were sometimes even used interchangeably, such as in [38]), some differences may be seen in the way these concepts were translated into actual

implementations. Among those examined, the applications that claimed to implement or enable context-*aware* communications often focused on the use of high-level user-related context (e.g. social situation) to influence how actual humans would conduct their interaction at a personal or social level; context-*sensitive* communications on the other hand were often used in connection with low-level, device-to-device or application-to-application interactions. There is no evidence however that the distinction was by design or in any way intentional in nature; it is merely presented here in an attempt to position the concept of context-tagged flows in relation to these synonymous concepts.

The concept of context-tagged flows seems to be more related to "context-sensitive communications," at least in the sense previously discussed, since this thesis is more concerned with the notion of flows at the level of interaction between network- and end-devices as well as applications, rather than at the level of personal conversation. However, it should be pointed out that one possible use of context tags would be to encode, preserve and convey, whenever useful or relevant, the high-level (personal, social) contextual and semantic aspects of a conversation that generate a network flow so that the flow may be appropriately handled within the network and by end-applications. In addition, in most cases both context-aware and context-sensitive communications often deal with the use of and adaptation to context by end-applications. The work in this thesis, in contrast, is more similar to the focus in the Ambient Networks (AN) project, where context is not only used by end-applications, but more importantly, by network entities as well [31].

A further examination of the nature and properties of CSCs in ambient networks (ANs) however reveals that in addition to some conceptual similarities with context-tagged flows, there are crucial differences as well. In ANs, CSCs consisted of conceptual channels needed to convey information as well as channels that conveyed control, context and meta-information governing the exchange. In context-tagged flows, there is the flow itself, which maps to the information channel in CSCs, and the context tags, which may be considered as the control/context channel in CSCs. However, in ANs, the entities of interest as far as context sensing and dissemination were concerned were quite general in nature ("network entities") [31]; in contrast, the approach in this thesis explicitly identifies flows as central entities of interest for context sensing and context-based network adaptation.

The approach of focusing on flows and developing the flow context concept has certain advantages over the other approaches reviewed in this chapter, such as the use of network-related context, network-centric context, context-aware communication, and context-sensitive communications. Some of these advantages are:

1. A flow is a well-defined and easily identified entity, making it easy to distinguish

the entity itself from its internal state, its situation and setting, and external environment, i.e., its context. In contrast, it was explained earlier that the same is not true for the more abstract idea of network context. As mentioned earlier, in ANs the contextual entities of interest were quite general in nature [31]. The definition of flow context on the other hand has operational usefulness because it can easily be translated into designs and architectures.

2. The notion of flow context does not implicitly exclude user context; in fact the definition asserts that it is an essential component of it. The concept of flow context therefore provides a more complete picture describing the interaction of the user with the network.

While the use of context to describe flows is novel and potentially useful, it should not be said that flow context alone would be sufficient to encompass all classes of context-aware networks, nor that the definition of flow context altogether replaces the ideas of network-centric or network-related context. Rather, the concept of flow context further refines these concepts and gives them a more concrete, tangible character. Flow context complements rather than competes with these concepts. While network context is broad and abstract, flow context is more well-defined in scope and concrete in nature. Finally, the work in this thesis also maps quite well within the space of, rather than competes with, the concept and vision of situated and autonomic communications, since it provides concrete conceptualizations and offers mechanisms for their realization.

3.6 Chapter summary

This chapter began by defining some of the key concepts used in this thesis, such as context and context-awareness, and argued for the adoption of an entity-centered approach to building context-awareness within networks, leading to a focus on flows as contextual entities of interest. Consequently, the concepts of flows, flow context, flow context tags, and context-tagged flows, were defined.

To further understand and appreciate the concept of flow context, some of its characteristics were described, such as its dynamism, imperfectness, its multiple representations, and relationships with other types of context. It was also mentioned that flow context may be intrinsic or extrinsic to the flow itself, and that it may be explicitly represented or exist implicitly within the environment, or in other pieces of information. The idea that flow context may exist implicitly, and that it might require some form of sensing, processing

and dissemination, before it can be used, led to the definition of the various stages of its notional life cycle.

Finally, related concepts in the literature, such as network-related context, network-centric context, context-aware communication, context-sensitive communications, and situated and autonomic communications were compared to the key concepts defined in this thesis. It was argued that the strategies of (1) focusing on flows as contextual entities, (2) defining and using flow context, and (3) endowing it with a broad and inclusive character, have their advantages over related concepts and approaches. However, the view is that the work described in this thesis occupies a distinct and novel solution space that can complement, rather than compete with, these similar visions.

Chapter 4

Modeling Flow Context

In the previous chapter, a notional life cycle for flow context was presented, describing the various stages that pieces of context information may go through before they can be used by interested consumers within the network, or by end-applications. Before such a life cycle can exist, a system that performs the corresponding functions has to be designed and implemented, and during the design process, the representation or *model* for the information to be exchanged and processed has to be determined.

Why is it important to develop a model for flow context? Perhaps the answer to this question may be found within the definition of the word itself. The Oxford English Dictionary[1] defines a *model* as:

> "a simplified or idealized description or conception of a particular system, situation, or process ... put forward as a basis for theoretical or empirical understanding, or for calculations, predictions, etc.; a conceptual or mental representation of something."

This definition offers two possible uses of a model: (1) as a tool for understanding, and (2) as an end-product that can be used for calculations or for other forms of processing.

Since flow context is a new concept, it is important to develop a model, since the modeling *process* may further progress the understanding of the concept, and the end-product or *artifact* (i.e. the model itself) could help others understand the conceptualization and thereby effectively communicate the view within the research domain. On the other hand,

[1] http://dictionary.oed.com

since the broader objective is to enable the realization of context-aware networks, developing a model for flow context that can be used during runtime by context-aware networks as an aid in managing or manipulating flow context during the various stages of its life cycle would be extremely useful.

This chapter therefore focuses on modeling flow context. The next section provides an overview of context modeling approaches found in the literature. Section 4.2 argues for the use of ontologies as models for flow context and related concepts. Section 4.3 describes the ontology-building process, including the technologies and tools used for this purpose. Finally, Section 4.4 describes related work, and Section 4.5 concludes this chapter.

4.1 An overview of context modeling approaches

The following presents an overview of context modeling approaches, based on existing classification schemes described primarily in the domain of context-aware applications.

Some typical data structures used to represent context information in context-aware systems were enumerated by Chen and Kotz in a survey of context-aware mobile computing research [12]. This enumeration had since been extended and presented as a taxonomy of context modeling approaches by Wang et al. [91], Strang and Linnhoff-Popien [92] and Balakrishnan et al. [93]. They classify modeling approaches primarily on the structure of the representation used, into key-value models, markup-based models, graphical models, object-oriented models, and logic-based models. Strang and Linnhoff-Popien as well as Balakrishnan et al. augmented this list with what they called ontology-based models, while Balakrishnan et al. further introduced a hybrid model. On the other hand, Razzaque, Dobson and Nixon provided a smaller set of modeling approaches: those based on set theory, directed graphs, first-order logic, and profiles [71]. The models they enumerated, along with a few examples of how these schemes may be used to describe flows and their context, are summarized below:

- *Key-value pair models*, as the name implies, provide contextual descriptions using attribute lists in a key-value[2] manner. Implicit in this classification is the association of that key-value pair to the entity being described, so a more complete name might be *entity-attribute-value* (EAV) model. As an example, the Session Description Protocol (SDP) describes multimedia sessions[3] using lines of text of the form ⟨type⟩=⟨value⟩,

[2] A key-value pair refers to a pair of associated strings, where one, called the key, serves to identify an item (typically for indexing or searching purposes), and the other describes the value of that item.

[3] RFC 2327 defines a multimedia session as "a set of multimedia senders and receivers and the data

where ⟨type⟩ are single characters and ⟨value⟩ are structured strings whose format depend on ⟨type⟩ [94]. An SDP description `m=video 49232 RTP/AVP 0` would denote a media stream containing video, transported over UDP port 49232 using the Realtime Transport Protocol – Audio/Video profile, with format RTP/AVP payload type 0 [95, 96].

- Models based on *set theory*, as described by Razzaque, Dobson and Nixon [71], seem to be a generalization of the entity-attribute-value model, supporting the clustering of multiple key-value pairs into either tuples or sets of vectors that described a situation based on multiple *cues* (abstracted information from sensors) [97, 65]. It is unclear however from [71] how these models formally mapped to set theory.

- *Profiles* [71] extend the entity-attribute-value models into hierarchies of one or more entity-attribute-value trees, where values may recursively be other entities [98]. Since the key-value model mentioned earlier also allows for pair recursion [12, 93], such as in the work by Schilit et al. [99], a separate class called *markup scheme models* is used to denote hierarchical schemes that use languages such as the Extensible Markup Language (XML) [100] or some other form of markup-based encoding [92, 93]. For example, HQML is an XML-based hierarchical QoS markup language that allows distributed multimedia applications on the World Wide Web to signal their QoS requirements to end-systems, middleware, and network devices [101].

- *Graphical models* such as in the Unified Modeling Language (UML) [102] and entity-relationship (ER) diagrams [103] represent context models in a graphical fashion. In the Ambient Networks project, for instance, a graphical conceptualization for various network-related context, such as QoS and cost, was presented [109]. However, it should be noted that in this example, as in others [32], the use of a graphical model was intended mostly as a design tool convenient for humans, and that the final machine-readable model would usually serialized in some other format, such as in XML or RDF/XML [104].

- *Object-oriented models* are based on the use of one or more principles of object-oriented design, such as the use of classes, class methods and messages, inheritance, encapsulation, abstraction, and polymorphism. For example, Serrano et al. used an object-oriented context information model for the management of pervasive network services, although the initial modeling was made using a scheme similar to entity-relationship diagrams [105].

- *Logic based models* describe context in terms of facts, expressions, and rules, and employ formal means by which other facts and expressions may be derived (reasoned

streams flowing from senders to receivers."

or inferred) from these. As an example, a context predicate describing the state of a network interface might be written as `Context(TrafficRate,InterfaceEth0,>,8 Mbps)` to describe the situation where the traffic on an interface exceeds 8 Mbps, using the `Context(⟨ContextType⟩,⟨Subject⟩,⟨Relater⟩,⟨Object⟩)` notation used in the *ConChat* and *Gaia* systems [83, 85].

- *Ontology based models* use a formal specification for contextual concepts and their attributes, restrictions, and relations. Such context models are typically expressed in a formal ontology language such as the OWL Web Ontology Language [106]. Some examples of recent work that fall under this category include the Context Ontology (CONON) by Wang et al. [91], CAMUS by Shehzad et al. [107], the Agent-Based Context-Aware Infrastructure (ACAI) by Khedr and Karmouch [89], ONTO-CONTEXT by Serrano et al. [108], and the work in Ambient Networks [109]. The approach used in this thesis also falls under this classification, and is compared in more detail with the other cited examples later in this chapter.

Although the taxonomy of models presented above is based primarily on the data structures used, structure alone is insufficient to classify the work found in the literature. For example, context modeled in an ontology language like OWL [110] may ultimately be serialized in RDF/XML, making it (according to the foregoing classification) a markup-based model. Thus mere format or structure alone does not distinguish one class of modeling from the next, and is insufficient basis for comparison or useful evaluation.

While the syntax and structure of context modeling schemes used in different systems are important, the ability of a modeling scheme to capture and explicitly represent semantics – *meaning* – is equally crucial, and adds another dimension to the classification and evaluation of models. One could perhaps distinguish or compare models also in terms of their semantic explicitness, formality, and expressiveness, that is, where they might lie on some conceptual "semantic spectrum" [111, 112]. The following section explains why the ability to explicitly encode semantics has made the use of ontologies a popular approach in modeling context, and why it is adopted in this thesis.

4.2 The case for ontologies

A major component of the work in this thesis concerns the development and use of an *ontology* of flows and their context. An ontology is an explicit formal specification of the terms in a domain and relations among them [113]. Gruber further defined an ontology

as a specification of a conceptualization, where the term "conceptualization" referred to an abstract, simplified view of the world, to be represented for some purpose [114]. Fensel et al. in turn defined a conceptualization as "an abstract model of some phenomenon in the world that identifies that phenomenon's relevant concepts" [115].

Although there are numerous definitions and usages of the word "ontology" – for example, Guarino and Giaretta identified at least seven interpretations of the word [116] – the following definition by Guarino seems to be the most appropriate for the work in this thesis:

> "... in its most prevalent use in AI, an ontology refers to an *engineering artifact*, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the *intended meaning* of the vocabulary words. This set of assumptions has usually the form of a first-order logical theory, where vocabulary words appear as unary or binary predicate names, respectively called concepts and relations. In the simplest case, an ontology describes the hierarchy of concepts related by subsumption relationships; in more sophisticated cases, suitable axioms are added in order to express other relationships between concepts and to constrain their interpretation." [117]

An ontology therefore is a formal and explicit specification of concepts, classes, objects, and other entities in the domain of interest, as well as their relations, properties, and their restrictions.

Why are ontologies relevant to the study of flow context? Before this can be answered, it is useful to examine first why ontologies are developed in different domains or disciplines. Chen et al. stated that ontologies are key requirements in context-aware systems because (1) they enable knowledge sharing in open and dynamic distributed systems, (2) well-defined declarative semantics in ontologies provide a means for intelligent agents to apply reasoning to context, and (3) they allow devices and agents not expressly designed to work together to interoperate [118]. Noy and McGuinness on the other hand listed the following general reasons for developing ontologies [119]:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit

- To analyze domain knowledge
- To separate domain knowledge from the operational knowledge

Each of these reasons may be viewed within the context of the specific objectives and work in this thesis as follows:

Sharing common understanding of the structure of information There are two dimensions to this aspect, namely, (1) achieving common understanding between people, and (2) an understanding between software agents. To make the use of flow context a reality within the broad perspective of the Internet, a common understanding among the different research and developer communities – groups of people who may be using different implementations – would be required. On the other hand, at the operational level, the same shared understanding must also exist among end-hosts, network devices and middleware so that flow context information existing in one network administrative domain would be used in a consistent fashion within that domain, and would be similarly interpreted in other domains.

Enabling domain knowledge reuse The notion of flow context is quite broad, encompassing information not only about the flow but also relevant information about the users and their activities, applications, devices, links, protocols and other entities that generate or affect the flow. In some cases, ontologies governing these related entities already exist, or are in the process of being defined by research communities. For example, the Foundation for Intelligent Physical Agents (FIPA) defined specifications for a device ontology [120] and for quality of service [121], while an ontology for wireless networks was proposed by Helin and Laukkanen [122]. The World Wide Web Consortium (W3C) Device Independence working group also has a recommendation called Composite Capability/Preference Profiles (CC/PP), which provides a description of device capabilities and user preferences that can be used to guide the adaptation of content presented to that device [123]. In cases like these it would be extremely useful and efficient to leverage the existing work done on these ontologies, instead of reinventing them. In addition, the collaborative nature of the work that goes into the development and standardization of some ontologies implies wide acceptance within the community, and ensures interoperability between implementations that use them.

Making domain assumptions explicit An ontology for flows and flow context would explicitly define the objects that would constitute "flow context," their semantics, and

relationships. For instance, one issue raised with the notion of "network context" in Section 3.1.2 was that it was not well-defined, making it difficult to translate the concept into implementations. An ontology for flows and their context would therefore help avoid this pitfall. In addition, having a flow context ontology promotes the design and implementation of systems where the domain assumptions about flow context are not hidden or embedded in program code, making them more difficult to understand, change, update or debug; rather, these are explicitly defined and documented. These explicit definitions are also useful for new users who must learn what terms in the domain mean [119].

Analyzing domain knowledge The construction of an ontology is an analytical and often iterative process that involves a thorough examination of the knowledge in that domain. Aside from the final output, the *process* itself leads to a better understanding of the domain. This process would involve the enumeration of important terms and concepts that are relevant to flows and their context, the definition of their classes and subclasses, their properties, restrictions, allowed values, and instances [119]. This would also include the analysis and reuse of existing ontologies related to the domain of interest, such as existing device ontologies (e.g. [120]), quality of service ontologies (e.g. [121]), and wireless network ontologies (e.g. [122]). Although the process necessarily involves analyzing large amounts of information and knowledge, it should be done within a well-defined scope, keeping in mind the target application, in order to keep the effort manageable.

Separating domain from operational knowledge An ontology would help decouple the knowledge *about* flow context from the knowledge about the processes and mechanisms that *operate on* flow context. The domain knowledge about flow context would cover the entities that constitute it, their properties, structure, attributes and relationships; while the operational knowledge would deal with the methods that can be performed on flow context in its nature as information that can be processed and manipulated. This separation would allow different approaches and algorithms for context sensing, processing and management even in other domains (such as in the handling of user context) to be investigated, objectively compared, evaluated and considered for application to the domain models for flow context. Conversely, this separation also implies that novel methods developed for handling flow context may possibly find application in the handling of context in other domains as well.

In the previous paragraph, although it was mentioned that one of the uses of an ontology would be to separate domain from operational knowledge, it is interesting to note that conversely, an ontology may be used to map operational concepts and their corresponding

implementation artifacts to domain concepts. Ocampo et al. [124] for example used a flow ontology as a guide in the design of software components (Java classes) for a context-based flow classifier. Therefore, the following should be added to the rationale for using ontologies, adapted in part from Jasper and Uschold [125]:

- As a common, implementation-neutral language that can specify how flow and flow context artifacts can be authored and translated to specific implementations and target languages, ensuring their interoperability and reuse; and
- As a specification for modeling the requirements and design of software components that deal with flows and their context, and that perform classification, adaptation, management or other operations based on them

4.3 Towards an ontology for flows and their context

The discussion in this section begins with a few words about its title, specifically the use of the word "towards." An ontology aims to capture and represent the concepts – the knowledge – about a certain domain of discourse, and represent a shared, common understanding of that domain. As such, it is often the product of an iterative and collaborative effort between several people working in the field. Thus, it is expected that the ontology for flows and flow context developed in this work – and most other ontologies for that matter – would continuously evolve, both because either the domain itself changes or evolves, or the collective understanding of that domain changes or evolves.

With this in mind, an ontology for flows and their context is presented, with the view of making two main contributions: (1) an ontology that in itself can be put to immediate use, and (2) the same ontology, this time representing a *starting point* that could be further expanded, modified, validated, critiqued or evaluated by others in the field. An ontology that focuses on flows and links contextual concepts and relationships to flows is entirely novel and would eventually need to evolve into a *shared and common understanding* of the domain. It is hoped that the discussion in this section provides enough insight to allow the use of the developed ontology in either of these two stated ways.

4.3.1 Technologies for the Semantic Web

Ideally, the discussion of the design and implementation of the ontology should begin with a discussion of the design methodology and requirements, prior to the selection of the tools

and technologies for its implementation. However, the presentation of the ontology later in this section cannot entirely be separated from the technology (i.e., language) used for its representation. In addition, it is also important to understand the motivation for selecting these tools and technologies.

Why use Semantic Web technologies?

The ontology developed in this thesis is represented using the Web Ontology Language (OWL), which is part of a stack of recommendations by the World Wide Web Consortium (W3C) for the *Semantic Web*. The idea of a Semantic Web was put forth by the inventor of the World Wide Web, Tim Berners-Lee, in describing his vision of a future Web where information can be shared, integrated and reused more easily through the use of machine-interpretable meaning, or semantics [126]. According to the W3C, it involves two aspects: (1) the use of common formats for the interchange of data, and (2) the use of languages to specify how the data relates to real objects.[4]

The principal technologies of the Semantic Web are built on the foundation of Uniform Resource Identifiers (URIs) [127], Extensible Markup Language (XML) [100], and XML namespaces [128], supporting the current components which are the Resource Description Framework (RDF) [129], the RDF Schema language [130] and the Web Ontology Language (OWL) [110, 131].[5] These technologies are designed as a layered set of specifications and may be mapped to a "layered cake" architecture, where the languages and representations, in order of increasing expressive power, are stacked on top of each other [132], as illustrated in Figure 4.1.

Why were Semantic Web technologies used implementing the flow context model? There were a number of reasons for this, including:

1. These technologies, although formally expressed by the W3C in the form of Recommendations, are typically the result of extensive work and consensus-building and eventually become adopted as standards by the community. It would therefore be advantageous from an interoperability viewpoint to use these standards as well
2. With widespread adoption, more tools become available, and
3. Although the Semantic Web is intended for end-applications and systems, these are *networked* systems, and consequently it would be reasonable to expect the use of the

[4] http://www.w3.org/2001/sw/

[5] Appendix D ("Overview of Semantic Web Technologies") provides an overview of these technologies.

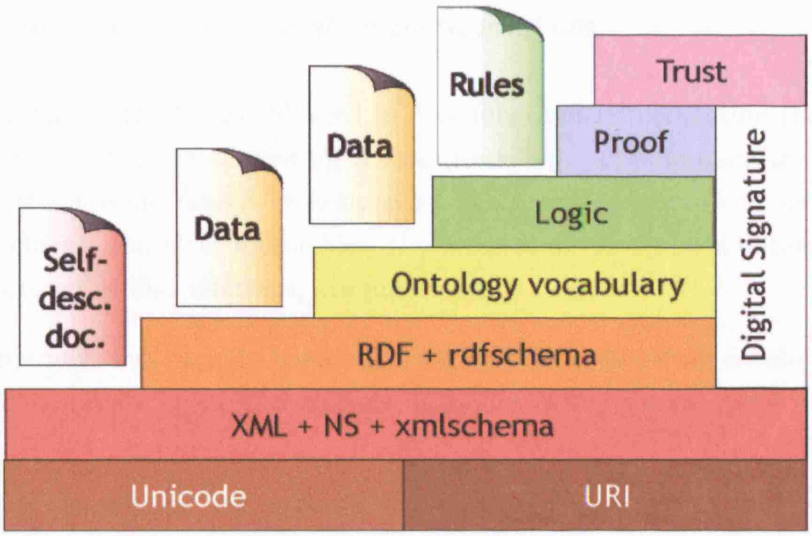


Figure 4.1: "Layered cake" architecture for the Semantic Web. From [132].

same technologies in network middleware and other network software.

The ontology layer in Figure 4.1 was selected for modeling in this thesis, as the immediate layer below it (represented by RDFS) was insufficient for this purpose. While RDF and RDFS allow the representation of *some* ontological knowledge, these are mainly concerned with the organization of vocabularies in typed hierarchies [292]. On the other hand, OWL (which instantiates the ontology layer in Figure 4.1) has a richer vocabulary that is able to further describe classes, relations between classes, and property types and characteristics [291]. For example, while RDFS only allows existing classes to be subclassed, OWL allows new classes to be constructed from existing ones, through enumeration, intersection, union, complement, and property restriction. OWL is designed to be a well-defined and Web-compatible ontology language, with supporting reasoning tools, a syntax intuitive to human users, formal semantics, and compatibility with existing Web standards. In addition, the ontology was implemented in the OWL DL sublanguage[6] because of its ability to provide expressiveness, while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time) [106]. Therefore, this was the language used in the flow context ontology, presented in the next section.

[6]See Appendix D: Overview of Semantic Web Technologies for more information on OWL DL.

4.3.2 Design methodology and implementation

In this section, the methodology followed in designing and implementing the ontology, as well as the tools used in implementing it, are described. The domain and scope of the ontology and the specific design objectives are discussed. Some of its salient features, including the classes and class hierarchies, the integration of external ontologies, and the modeling of properties and relations, are presented.

The design approach was roughly based on a method for engineering ontologies suggested by Noy and McGuinness in [119], consisting of the following general steps:

- Determine the domain and scope of the ontology
- Consider reusing existing ontologies
- Enumerate important terms in the ontology
- Define the classes and class hierarchy
- Define the properties of classes – slots
- Define the facets of the slots
- Create instances

The development process was based on these general steps, but not strictly in this order. In addition, it was also iterative in nature as expected [119]. To provide a better feel for this process, a few specific examples are given below:

- *Domain and scope of the ontology* - Includes concepts associated with flows of protocol data units (PDUs) through computer networks, and their contextual descriptions
- *Enumerate important terms in the ontology* - Flows, "layer-3" (network-layer) flows, packet count
- *Consider reusing existing ontologies* - Work in IPFIX (cf. page 77) can provide similar terms and definitions related to traffic characteristics.
- *Define classes and class hierarchy* - From IPFIX, the class `packetTotalCount` is defined, which is a subclass of `PerFlowCounter`, which in turn is a subclass of `InformationElement` (see Figure 4.4 on page 80).

- *Define the class properties* - `packetTotalCount` has the following properties: `dataType`, `dataTypeSemantics`, `elementID`, and `units`.
- *Define the property restrictions* - `elementID` can only take on `int` values; `units` can take on `string` values. For `packetTotalCount`, the `unit` value is `packets`.

A more detailed discussion on each of these steps is given in the next few sections of this chapter.

Domain and scope of the ontology

The domain of the ontology covers the fields of networking in general, and context-aware networks in particular. In addition, to provide both a conceptual and concrete links between the domains of 'conventional' flow classification and characterization on one hand, and context-aware computing on the other, related fields in the area of context-aware computing that intersect with the scope of the ontology's domain were taken into consideration.

The applicable domain of ontologies may also be illustrated through examples of their use [119]. In addition to the general uses mentioned in Section 4.2, its specific uses are:

- To standardize the definitions of flows and flow context concepts, and where standard definitions exist, to capture and encode these definitions in a formal, explicit and unambiguous way, particularly for humans and organizations.
- To serve as a guide in designing software agents that handle flows and detect or process their context.
- To encode the semantic basis for flow and flow context processing by software agents, network middleware, network protocols, end-host applications, reasoners, and other program code.

Table 4.1 further illustrates, through a concrete example, how the ontology might be used in the domain of QoS management.

Specific design objectives

In addition to the general reasons for creating ontologies (cf. Section 4.2), the following specific design objectives served as a guide in developing the ontology:

Ontology role	Specific example
Standardize definitions and terminology	Two service providers outline the terms of a QoS service level agreement (SLA)
Design guide in the implementation of software agents	Network administrators develop or deploy software that would monitor and enforce the terms of the QoS SLA
Provide semantic basis for flow processing	Software agents previously deployed detect and process flows, sense their context, and automatically enforce the terms of the SLA

Table 4.1: Example use-case for a flow context ontology in QoS SLA management.

1. *The ontology should provide a taxonomy of flows.* Section 3.2 showed that a wide variety of definitions and conceptualizations for the term 'flow' exist in the literature. This presents a problem, for instance, when two different entities need to exchange information or transact on the basis of some flow definition. To account for this, the basic concept of flows ("a sequence of related protocol data units") was formulated in the most broad and general terms as possible. The ontology then provided a scheme for specifying and classifying the different kinds, or *subclasses* of flows under this broad concept. This enables both humans and applications to distinguish one class of flows from another, provides for consistent terminologies, and serves as a guide in the design of applications intended to classify streams of protocol data units into flows.

2. *The ontology should describe the concepts of, and establish the links between, intrinsic flow context and extrinsic flow context.* At one conceptual level, the ontology should provide the link between the internal properties and characteristics of flows on one hand, and the external entities that form the flow's extrinsic context on another hand. In parallel with this, the ontology should likewise provide a link between the concepts and terms found in the 'traditional' domains of flow classification, metering and quality of service (QoS), where much of the concepts of intrinsic context may be found, with the concepts, terms and approaches in the relatively newer fields of context-aware computing and context-aware networks, where much work has been done on describing the context of users, devices and applications.

3. *The ontology should be able to link with existing related ontologies.* The ontology should, whenever possible, either directly import, or otherwise provide 'hooks' to existing related ontologies. This ensures that the ontology conforms with and is able to use the terms and conceptualizations that are known to be accepted widely within the domain of the linked ontology. In addition, this approach is practical from the point of view of efficiency, to avoid "reinventing the wheel." However, in some cases the existing ontologies might not provide the necessary concepts, or the level of detail required, or fail to express the necessary relationships with flow concepts. In such

cases some additional modeling needs to be introduced. Thus, while the immediate domain and scope of the ontology in this work relates to flows and networks, part of the development process would necessarily include the expansion or adaptation of concepts and relationships for contextual concepts that are normally centered on other entities such as humans, devices, and applications.

Finally, an additional motivation for ontology interoperability stems from the fact that not only is the ability to import related ontologies desirable: in the future, it would equally be useful for the ontology described in this thesis to be easily examined, processed, and imported by other people and applications in the domain.

Implementation tools

The use of Semantic Web technologies, particularly XML, RDF and OWL, is a direct result of the interoperability objective mentioned in the preceding section. Although several ontology authoring tools exist [133], the Protégé-OWL Plugin [134] open-source ontology modeling environment was selected and used to edit and maintain the ontology in this thesis, as it is freely available, has excellent support for OWL-DL and for a back-end reasoner, is actively in development and supported by developers, has a very well-written tutorial [135] available, and has a relatively large and active community of users.

Finally, ontology checking and reasoning services were provided by a software package called RacerPro [136]. RacerPro can easily be integrated with Protégé, has a rich built-in query/rule language called nRQL supporting queries over knowledge bases, and is well-documented. Although RacerPro is commercial software, a free educational license was provided in support of the work described here.

Classes and class hierarchies

This section describes the structure of the ontology[7] for flows and flow context in terms of the defined classes and the class-subclass hierarchies.

Top-level concepts Two main classes called `DomainConcepts` and `LocalConcepts` are immediately defined below the built-in top-level class `owl:Thing`.

`DomainConcepts` refer to concepts that have official or generally-accepted definitions, or

[7] Available at http://www.ee.ucl.ac.uk/~rocampo/flow.owl

those that are likely to be adopted by the community. Sub-concept definitions and properties under `DomainConcepts` are typically based on documents such as Request for Comments (RFCs) and other Internet standards, and may import or adopt concepts from existing ontologies. Subclasses (sub-concepts) directly under `DomainConcepts` include:

- `Flow` is a key concept defined in the ontology to provide a root class for a taxonomy of the different types of flows. `Flow` and its sub-concepts are linked via OWL properties to other sections of the ontology that describe flow context concepts. The structure and subclasses under `Flow` will be discussed shortly under the section on "Flow Taxonomy" on page 72.

- `FlowTrafficCharacteristic`, `FlowTimeCharacteristic`, `FlowContentType`, as well as `FlowContentEncoding`, `FlowDirectionality`, `FlowFanout` and `FlowPDU_Info` are general concepts under which the various subconcepts describing intrinsic flow context are defined. These concepts will be further described shortly under the section on "Intrinsic Flow Context" on page 74.

- `NetworkMedium`, `NetworkedEntity` and `User` are some of the main classes that subsume concepts related to extrinsic flow context. The properties of networked entities are described under the class `NetworkNodeCharacteristic`. Classes related to extrinsic flow context are further described under the section on "Extrinsic Flow Context" on page 76.

- The `OSI_Layer` encapsulates the layers of the OSI Reference Model [137, 138], providing a means to map protocols and flows to its different layers. As a naming convention, and only when feasible, concepts in the ontology are prefixed with tags that identify the layers to which they are mapped. For example, the class of protocols that map to OSI Layer 4 (the Transport Layer) is named `L4_Protocol`, and an example of a sub-class under this is `L4_UserDatagramProtocol`. These prefixes are simply for presentation purposes and do not have any semantic value (that is, to a machine).

- Protocol-related concepts such as `RoutingProtocol` and `TransmissionProtocol` provide top-level concepts for the various classes of communication protocols that govern routing and transmission, respectively. The behavior and states of protocols are classified under `ProtocolState`. `ProtocolDataUnit` contains classes for the different types of PDUs handled by the different protocols. For example, for network-layer flows that are governed by IP, the `L3_IPv4PDU` class defines an IP version 4 packet. On the other hand, one example of an application-layer PDU might be an `HTTP_ProtocolMessage`, which further subsumes HTTP request message classes

(connect, delete, get, head, options, post, put, and trace) and HTTP response messages [26]. `FlowPDU_Info` provides a conceptual link between flows and their PDUs, by describing the attributes or properties of flows that are directly encoded in the fields of their PDUs.[8] These fields, as well as other structural components of PDUs are described under the class `ProtocolDataUnit_OR_Component`.[9]

- The `Identifier` class encodes concepts related to the various identifiers for entities and resources used in networks. Subclasses under `NetworkedEntityIdentifier` are organized by OSI layer and include concepts such as IP addresses, network prefixes and netmasks [139, 140]; Internet DNS names [141], and uniform resource identifiers (URIs) [127]. On the other hand the `FlowIdentifier` sub-class under `Identifier` provides for flow identifier concepts such as the *filterspec* used in the Resource Reservation Protocol (RSVP) [142], or flow labels in Multiprotocol Label Switching (MPLS) [143] and in IPv6 [140]. Various end-applications or operating systems provide means to identify flows through the use of filter constructs like the u32 filter in Linux [144], or rules such as those used in intrusion detection systems (e.g. Snort [55]) and in various email spam filters [54].

The main classes under `DomainConcepts`, as edited in Protégé and displayed graphically via the OWLViz tool are shown in Fig. 4.2. Note that all other subclasses have been suppressed from the display for presentation purposes.

Finally, as a counterpart for `DomainConcepts`, `LocalConcepts` encapsulates classes that would tend to have more limited scope of application or adoption since they may depend on subjective assessments or value judgments within a local (e.g. administrative) domain. For example, value partitions such as `FlowThreatLevel` (containing the subclasses `Safe`, `Suspicious` and `Malicious`) are contained in this class.

Flow taxonomy As previously mentioned, the generic `Flow` concept roots a tree that provides a taxonomy of flows. The main subclasses are organized according to their mapping to layers in the OSI Model, resulting in the classes `Layer2Flow` (link-layer flows), `Layer3Flow` (network-layer flows), `Layer4Flow` (transport-layer flows), and `Layer7Flow` (application-layer flows). Based on the definition of the flow concept, each flow class describes a related sequence of PDUs that exist at that layer. Thus, a sequence of (related)

[8] Note that a modeling distinction is made between the *fields* of the PDUs and the *information* within these fields. For example, the source IP address field of an IPv4 packet, and the source IP address information encoded in that field, are considered to be two distinct entities.

[9] The rather curious name for this class will be explained later in this section, under "Part-Whole Relations" on page 86

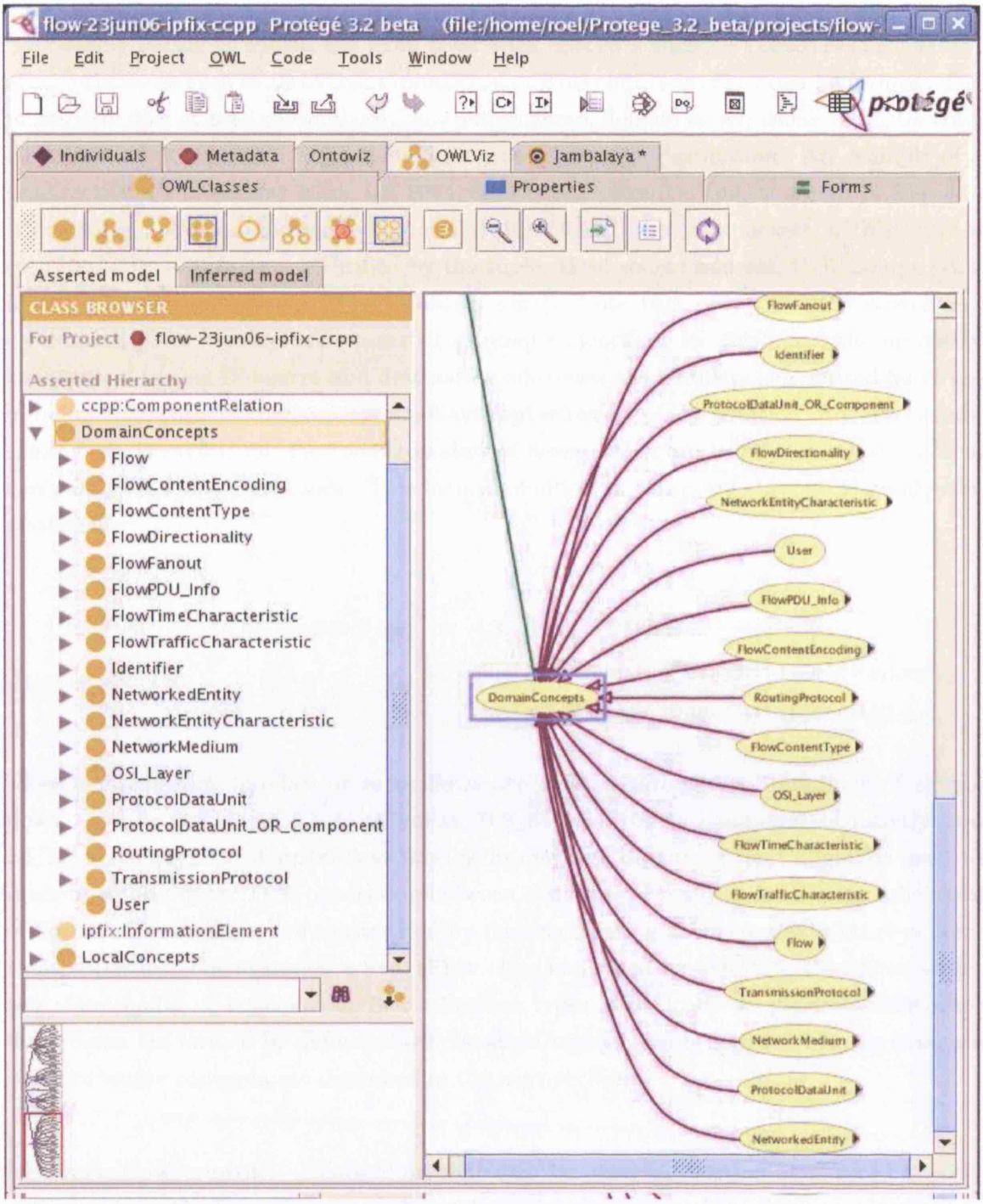


Figure 4.2: The main ontology concepts under `DomainConcepts`, as seen in Protégé

Ethernet frames would be considered an instance of a `Layer2Flow`, a sequence of IP packets an instance of a `Layer3Flow`, or a sequence of HTTP messages an instance of `Layer7Flow`.

By relating concepts within the `Flow` class with concepts such as `FlowDirectionality`, other subclasses such as `UnidirectionalFlow` or `BidirectionalFlow` can be formed. The most basic flow classes are unidirectional simple flows, defined as sequences of PDUs traveling in a single direction, from a single source to a single destination. An example of a unidirectional simple flow class, `L3_IPv4_OneWay_UDP_SimpleFlow`, is shown in Fig. 4.3. It describes a class of unidirectional flows of (layer-3) IP version 4 packets, with IP protocol type UDP, and further identified by the tuple ⟨IPv4 source address, UDP source port, IPv4 destination address, UDP destination port⟩. Note that only the UDP source and destination ports are explicitly asserted as unique identifiers for this class; the necessary condition of having IP source and destination addresses as identifiers is inherited by virtue of being a subclass of the `L3_IPv4_OneWaySimpleFlow` class. In addition, it is also a subclass of `L3_IPv4PDU_UDP_PayloadFlow` class of flows, which are IPv4 flows whose packets carry only UDP-type payloads. This latter definition is expressed through the following statement:

$$\begin{aligned} \mathsf{L3\_IPv4PDU\_UDP\_PayloadFlow} \equiv\ & \mathsf{L3\_IPv4\_PDUFlow} \\ & \sqcap(\exists\, \mathsf{hasPDU.L3\_IPv4PDU\_UDP\_Payload}) \\ & \sqcap(\forall\, \mathsf{hasPDU.L3\_IPv4PDU\_UDP\_Payload}) \end{aligned}$$

More complex flow bundles or macroflows are defined through combinations of simple flows. For example, an `L3_IPv4TwoWay_TCP_SimpleFlow` is composed of exactly two `L3_IPv4_OneWay_TCP_SimpleFlows` flowing in opposite directions, and might be used to describe a full-duplex TCP connection between two hosts. Finally, more diverse subclasses of flows and macroflows are constructed by relating existing flow types with other contextual properties. For example, a `MediaFlow` class is defined by asserting that flows under this class should be transporting `MediaContent` types of payloads. In general, these other flow classes are formed by defining flow classes in terms of both their intrinsic or extrinsic context, whose concepts are discussed in the next section.

Intrinsic flow context Information that may be observed, sensed or inferred directly within or from the flow, describing the intrinsic characteristics of the flow, are classified as *intrinsic flow context.* `FlowTrafficCharacteristic` contains concepts that describe the QoS characteristics of flows, such as traffic rates, delay, jitter, and other metrics, under the

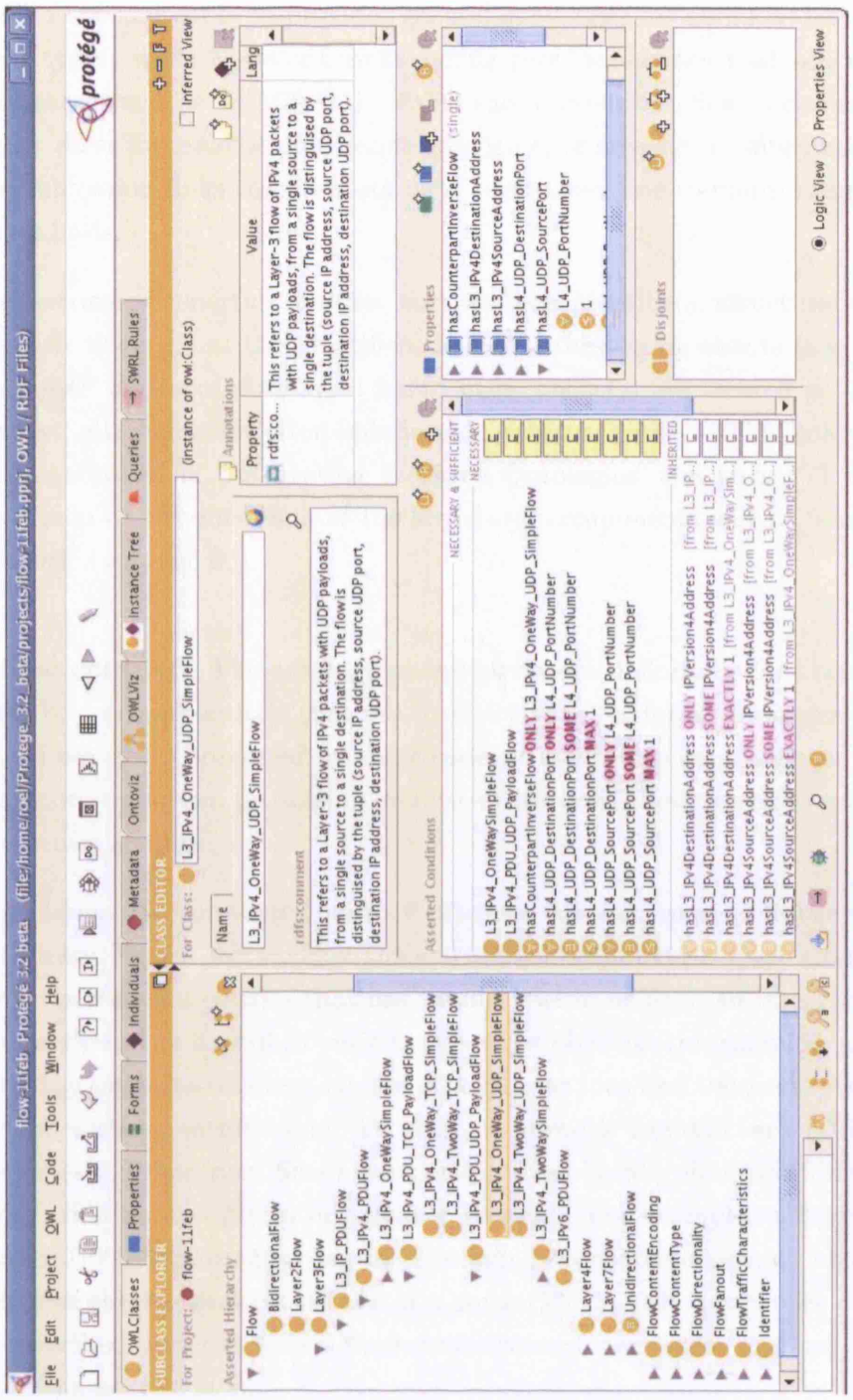


Figure 4.3: A unidirectional simple flow class in Protégé

sub-class `FlowQoSDescriptor`. `FlowTrafficClass` classifies flows along more application-oriented and qualitative descriptions such as bulk, interactive, streaming or transactional traffic classes. `FlowContentType` provides general subclasses for audio, video, mixed, and other content types, while `FlowContentEncoding` provides classes that describe content encoding schemes, such as MIME [21]. `FlowFanout` describes flows according to their scope, whether these are unicast, multicast, broadcast, or anycast, or alternatively, if they provide communication links to end-hosts on a one-to-one, one-to-many, many-to-one, or many-to-many basis.

Several other intrinsic properties of flows, such as their identifiers, structural components (PDUs and their fields), and the identifiers found in these components (e.g. the source and destination IP address information found in IP packets) are treated as part of their intrinsic context, although they often link flows to external entities. This link with various identifiers is established in "Integrating External Ontologies" (on page 77), and a more detailed discussion on the modeling of the structural components will be found in "Part-Whole Relations" (on page 86).

Extrinsic flow context The ontology includes concept definitions for classes of physical, electronic or software entities that use, connect with, or form an integral part of the network. Since entities represented by these concept definitions can influence the generation, transmission, reception, processing and use of the flow, these concepts are considered part of its extrinsic context.

Three major classes are considered part of extrinsic context, namely `NetworkedEntity`, `NetworkMedium` and `Users`. `NetworkMedium` and `NetworkedEntity` contain classes for physical, electronic or software entities that use, connect with, or form an integral part of the network. `NetworkMedium` describes various classes of physical transmission media, while `NetworkedEntity` contains different classes of `Applications` and `Devices`. As mentioned in the previous section, members of the class of network `Identifiers` (URIs, domain names, IP addresses, other identifiers at various protocol layers) often provide associations between flow instances and pieces of extrinsic context. For example, a flow may be related by `hasL3_IPv4EndpointAddress` to a certain `IPVersion4Address`, which, in turn, is associated with the `NetworkInterface` of a `NetworkNode`. The properties of networked entities are described under the class `NetworkNodeCharacteristic`, and are discussed in more detail in the next section.

The `User` class serves as a link to the relatively large body of concepts available from the work (by others) in context-aware computing. Rather than define additional concepts such

as, for instance, `UserActivity` or `Location`, these are intended to be imported from other relevant ontologies, such as those discussed in Section 4.4. Although these are not explicitly represented in the ontology presented in this thesis, they still form part of extrinsic flow context.

Integrating External Ontologies

An important step in the development of the ontology in this thesis concerns integration with external ontologies. This is a recommended approach that should be considered when possible [119], especially if the ontology represents the result of mature and collaborative work among experts in the domain.

There are actually a number of different ways that ontologies may be "integrated" and effectively reused. Pinto et al. identified three different meanings of the term, namely, (1) the reuse or importation of concepts from existing ontologies to supplement other concepts in a new one, (2) the merging of existing ontologies about the same subject to form a single unified ontology, and (3) the use of an ontology in a specific application [145]. Relationships between ontologies may be further classified as [146]:

- *Extensions between ontologies.* Ontologies are extended by the inclusion of concepts from other ontologies, or by importation of entire external ontologies
- *Identity and equivalence between ontologies.* Ontologies are identical if they are mirror equivalents, except possibly for their names. Ontologies are equivalent if the share the same vocabulary and logical axiomatization, but are expressed using different representation languages
- *Translation between ontologies.* A source ontology may be translated into a destination ontology by expressing the former in the representation language used by the latter. Ontologies may be *weakly translatable* if there is some loss of information during the translation process, and *strongly translatable* if the vocabulary and logical axiomatization are preserved, there is no information loss, and no semantic inconsistency is introduced.

In this section, and in the one that follows, examples of the importation or reuse of concepts from external ontologies (i.e. the first case in Pinto et al. above), but which require some degree of ontology translation, are described. An attempt to represent the concepts developed in the IP Flow Information Export (IPFIX) Working Group of the Internet

Engineering Task Force and the subsequent importation of these concepts into the flow context ontology is presented. In the subsequent section on "Properties and Relations" (on page 81) the same process is performed on concepts developed for the Composite Capability/Preferences Profiles (CC/PP) and the User Agent Profile (UAProf) specifications of the World Wide Web Consortium (W3C) and the Open Mobile Alliance (OMA), respectively.

The IPFIX Working Group's goal is to "produce standards-track RFCs describing the IPFIX information model and export protocol RFCs" for applications that require flow-based IP traffic measurements [47]. The information model is currently under development and exists as an Internet-Draft [147], with a detailed textual description of the elements of the model as well as an XML-based specification of these information elements and the abstract data types.

Although the IPFIX Internet-Draft provides textual definitions and an XML schema for the information model, it has no formal ontological representation written in RDF/XML or OWL. While the XML schema representation might be sufficient for the goals of the IPFIX WG – that is, to provide a standard *syntax* for describing flow information – such a representation cannot be directly imported into an ontology written in OWL-DL such the one developed in this thesis. In addition, the native XML schema in IPFIX cannot represent the relationship between the information in that model and the concepts defined in the flow ontology described here.

Thus, using the IPFIX Working Group's (draft) model, an equivalent model in OWL-DL was constructed.[10] As will be seen in the succeeding discussion, some aspects of the IPFIX model could not be faithfully replicated due to some weaknesses in the OWL-DL language. However, the OWL-DL model sufficiently reflected the main concepts in IPFIX and was successfully integrated with the flow context ontology. The implementation is discussed in the next few paragraphs. In addition, the prefix `ipfix:` is to refer to the OWL-DL version of the IPFIX model.

A root concept called `ipfix:InformationElement` was created and 12 subclasses were defined under it, based on the logical groupings in the IPFIX model. Note that these subclasses, as enumerated below, are not formal elements in the IPFIX model, but were artificially created as classes in the ontology to provide some logical structure. The subclasses and their respective definitions (as taken from [147]) are:

- `Identifier` - this class contains identifying components of the IPFIX architecture, of an IPFIX device, or the IPFIX protocol

[10] Available at http://www.ee.ucl.ac.uk/~rocampo/ipfix.owl

- `MeteringAndExportingProcessConfig` - this class contains concepts that describe the flow metering process or the flow record exporting process
- `MeteringExportingProcessStat` - this class contains concepts that describe statistics of the flow metering process or the flow record exporting process
- `IpHeaderField` - this class contains concepts that indicate values of IP header fields or values that are derived from IP header field values in combination with other information
- `TransportHeaderField` - this class contains concepts related to transport header fields and length
- `SubIPHeaderField` - this class contains concepts related to what are called "Sub-IP header fields" in the IPFIX model, referring to fields found in lower-layer frames such as Ethernet source and destination MAC fields.
- `DerivedPacketProperties` - this class contains concepts related to values derived from other values found in header fields. For example, the next-hop address of a packet in a flow may or may not be directly encoded in the packet header, but it may be derived nonetheless for instance by a router by examining the destination IP address of the packet and looking up the appropriate address from its routing table.
- `MinMaxFlowProperties` - concepts in this class describe values that are obtained by taking the minimum or maximum of certain values over packets from the entire flow, such as the minimum and maximum packet lengths.
- `FlowTimeStamp` - concepts in this class describe time stamps for flow events
- `PerFlowCounter` - concepts in this class describe counters that contain flow properties that potentially change each time a packet belonging to the flow is observed
- `MiscFlowProperty` - concepts in this class describe flow properties pertaining to the start, duration and termination of flows, but which are not time stamps
- `Padding` - this class contains a single element class which is used to pad flow records. It is included in the ontology simply for completeness.

A detailed discussion of each information element in IPFIX is beyond the scope of this thesis. However, for further information, and for the latest version of the information model, the reader is kindly referred to the website of the IPFIX Working Group, at http://www.ietf.org/html.charters/ipfix-charter.html.[11]

[11] It should be emphasized that the preliminary `ipfix` ontology developed here was based on work in progress in the IPFIX WG, and thus should not be considered to be based on a stable standard.

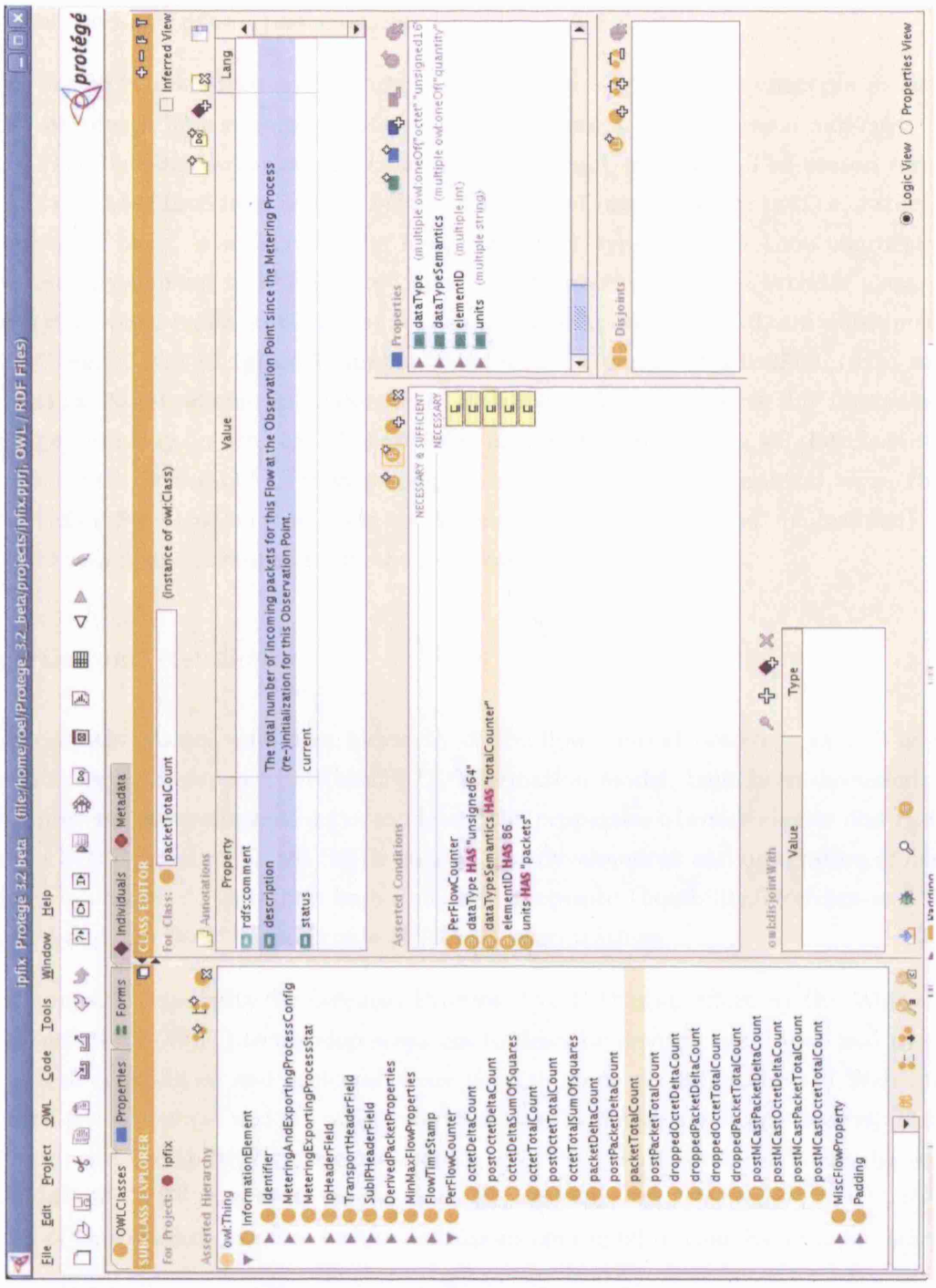


Figure 4.4: A class description in an ontology based on the draft IPFIX information model

A snapshot of the `ipfix` ontology is shown in Figure 4.4. This ontology was imported into the flow context ontology (henceforth called the "main" ontology) using an `owl:imports` statement, and an `ipfix:` prefix was used to refer to its classes.

Classes of `ipfix` are subsumed as subclasses under the appropriate concepts in the flow context ontology. For example, `ipfix:FlowTimeStamp` is mapped as a sub-class under the `L3_FlowTimeCharacteristics` concept in the main ontology. The reason for mapping `ipfix:FlowTimeStamp` as a sub-class of `L3_FlowTimeCharacteristics`, rather than an equivalent class, is to convey the idea that other types of flow time characteristics not necessarily defined in IPFIX may later be subsumed under this broader class. Similarly, `ipfix:DerivedPacketProperties` and `ipfix:IpHeaderField` are subsumed under `L3_FlowPDU_Info`; `ipfix:TransportHeaderField` under `L4_FlowPDU_Info`; and finally `ipfix:PerFlowCounter` under `L3_FlowQoSDescriptor`. Figure 4.5 illustrates the integrated ontology in Protégé, including a partial graphical view of the class hierarchies displayed by the OWLViz visualization tool. In the latter graphical view, the imported `ipfix:PerFlowCounter` class and its subclasses are "attached" (subsumed) under the `L3_FlowQoSDescriptor` class in the main ontology.

Properties and Relations

So far, only the classes and class hierarchy of the flow context ontology, as well as those in the ontology developed from the IPFIX information model, have been discussed. This section presents some approaches in modeling the properties of these classes and the relationships between them. To aid the discussion, the development and integration of another ontology is presented, this time based on the Composite Capability/Preferences Profiles (CC/PP) and the User Agent Profile (UAProf) specifications.

The Composite Capability/Preferences Profiles (CC/PP) is an effort by the World Wide Web Consortium (W3C) to develop a system to describe device capabilities and user profiles. These capabilities and preferences are used to guide the adaptation of Web content presented to the device, and are often referred to as the device's *delivery context.* Through the Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies Recommendation[12] [123], the W3C defines both the structure and vocabulary of CC/PP, and the rest of this discussion relies on the definitions contained in that Recommendation.

A *CC/PP profile* is essentially a two-level hierarchy, with each profile having at least one or more *components*, with each component having at least one or more *attributes.* Some

[12]Referred to as "Recommendation" in the rest of this section.

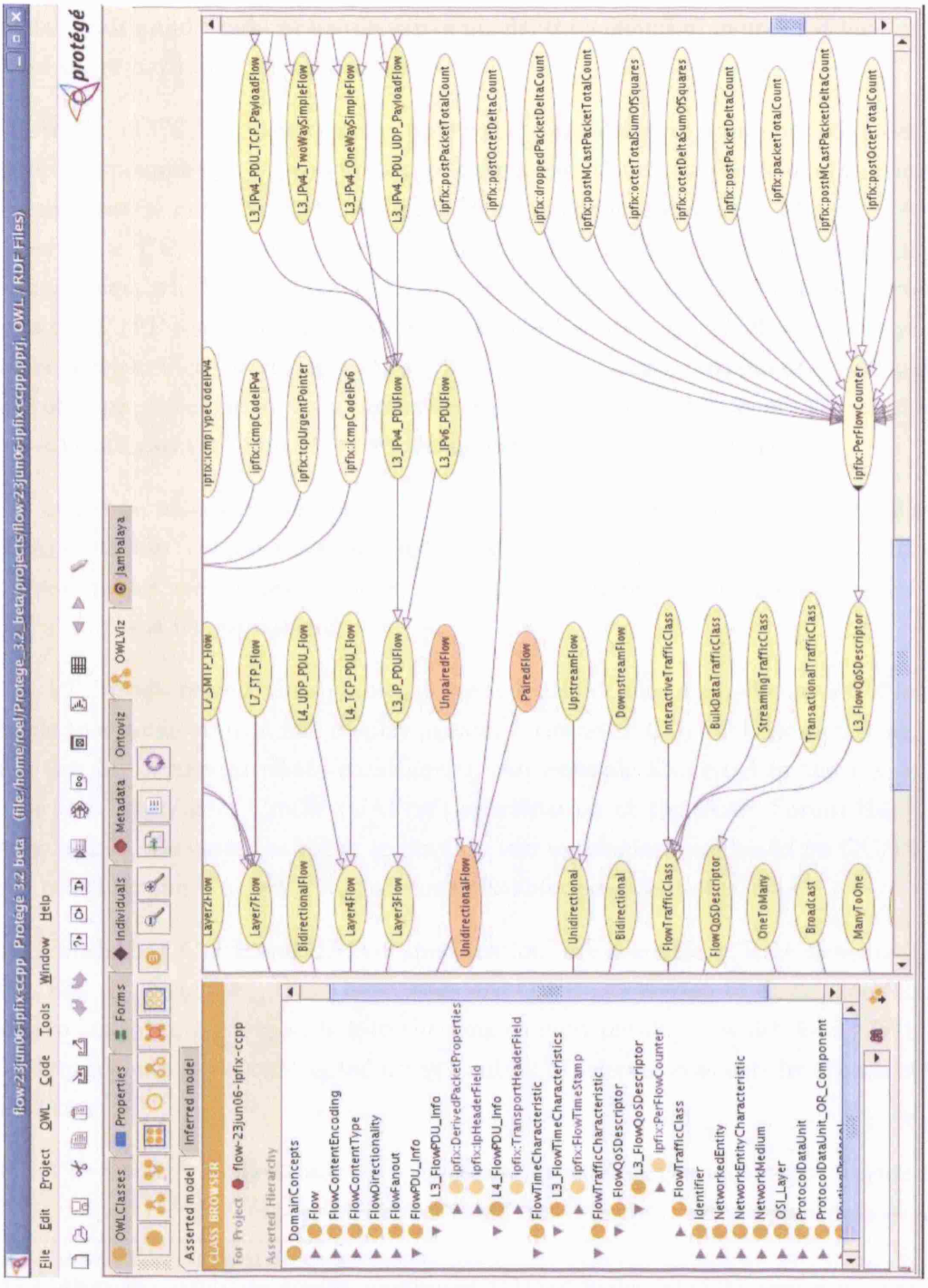


Figure 4.5: Integrating the `ipfix` ontology with the flow context ontology

examples of components would be the hardware platform, the software platform, or an application executing on a wireless device. The attributes of the hardware platform might include the width and height of its display in pixels, the amount of memory it has, its CPU type, and other hardware characteristics.

More formally, in CC/PP, a resource `ccpp:Profile` is related to a number of resources of type `ccpp:Component` by the property `ccpp:component`, that is, the domain of the property `ccpp:component` is `ccpp:Profile` while its range is `ccpp:Component`. The root class for all properties in CC/PP is called `ccpp:Property`. This class has two subclasses, namely `ccpp:Structure`, which is the class of all properties describing relations between structural elements of CC/PP itself, and `ccpp:Attribute`, which is the class of all properties relating components with their attribute values. The property `ccpp:component` is actually an instance of `ccpp:Structure`, while properties such as (for example) `ccpp-client:charset` or `ccpp-client:deviceIdentifier` would be instances of `ccpp:Attribute`.

CC/PP attribute values are either RDF plain or typed literal values, sets of values, or sequences of values. Some recommended data types (i.e., lexical space constraints) for plain literal values are strings, integers, and rational numbers, although the use of other valid RDF forms is not prohibited.

The CC/PP Structure and Vocabularies document also defines a small core set of features applicable to a range of print and display agents.[13] However, CC/PP is extended primarily through the use of new *attribute vocabularies*. An example also cited in the Recommendation is the User Agent Profile (UAProf) specification of the WAP Forum [148]. The following section discusses the effort to develop two ontologies: one based on CC/PP, and another based on the UAProf V2.0 Approved Enabler specification [149].[14]

Although both the CC/PP and UAProf specifications are available as RDF Schema (hence, in OWL Full, cf. page 248), there is no standard OWL-DL version of these specifications. In order to integrate both models into the flow context ontology (which is in OWL-DL), OWL-DL ontologies were constructed using a subset of selected concepts from each of these specifications.[15]

The CC/PP ontology defines only two concepts: `ccpp:Profile` and `ccpp:Component`.[16] An OWL property `ccpp:component` relates these two concepts, with `ccpp:Profile` as the

[13] http://www.w3.org/2002/11/08-ccpp-client#

[14] Unless otherwise specified, subsequent mentions of "UAProf" in the rest of this section are in reference to this particular version.

[15] Available at http://www.ee.ucl.ac.uk/~rocampo/ccpp.owl

[16] The prefixes `ccpp:` and `uaprof:` as used in this ontology refer to internal namespaces used for experimental purposes, and not the namespaces for the standard specifications. In fact, UAProf actually requires the use of the prefix `prf:` to refer to the official UAProf namespace [150].

domain and `ccpp:Component` as its range. A separate UAProf ontology that imported the CC/PP ontology was created, and the subclasses `HardwarePlatform`, `SoftwarePlatform`, `BrowserUA`, `NetworkCharacteristics`, `WapCharacteristics` and `PushCharacteristics` were subsumed under `ccpp:Component`.

The next step would have been the creation of the attributes defined in [149], such as `BitsPerPixel`, and their association with their respective components by defining their domain (i.e., `HardwarePlatform` in the case of `BitsPerPixel`). However, the UAProf attributes, while properties themselves, also have properties of their own: each attribute has a datatype such as "Boolean," "Number," "Literal" and "Dimension." In addition, each UAProf attribute has a property called a *resolution rule* that defines the appropriate action an application should take if conflicts arise when merging an existing profile with a new one, either by (1) `Override`(-ing) the old attribute value with the new values, or (2) `Lock`(-ing) the old attribute and effectively ignoring the new one, or (3) `Append`(-ing) the new values to the old values, typically in a list [150].

While it would have been relatively simple in OWL to define an attribute that relates a component to its value using an `owl:DatatypeProperty`, further describing this attribute in terms of its resolution rule presented somewhat of a problem because properties in OWL and RDF are binary relations (as will be discussed further in the next section). A possible solution might have be to describe the resolution rule as an annotation to the UAProf attribute, using an `owl:AnnotationProperty` [151]. However, in OWL DL, annotation properties cannot have any restrictions such as cardinality or domain/range restrictions, and reasoners will not use the annotation information for reasoning [152]. To avoid these limitations, an alternative solution was adopted, by modeling UAProf attributes as *n-ary relations*. This is discussed in the next section.

N-ary relations The issue of modeling properties of properties in OWL, without resorting to annotation properties, falls under the scope of what are known as *n-ary relations* [153]. This term generally refers to relations that link an individual to more than one individual or value.

The problem is that in OWL and RDF, properties are *binary relations* linking two individuals or an individual and a value [129, 151]. The issue at hand is how to describe additional properties, such as the resolution rule of a UAProf attribute, when the attribute is also a property in itself. This is illustrated in Figure 4.6, using the `BitsPerPixel` attribute as an example. An instance of `HardwarePlatform`, `HardwarePlatform_A`, has a `BitsPerPixel` value of "8." The `BitsPerPixel` attribute itself has a resolution rule property with value

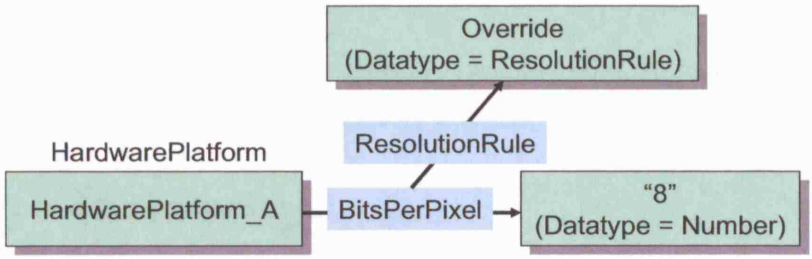


Figure 4.6: An n-ary relation in a UAProf attribute

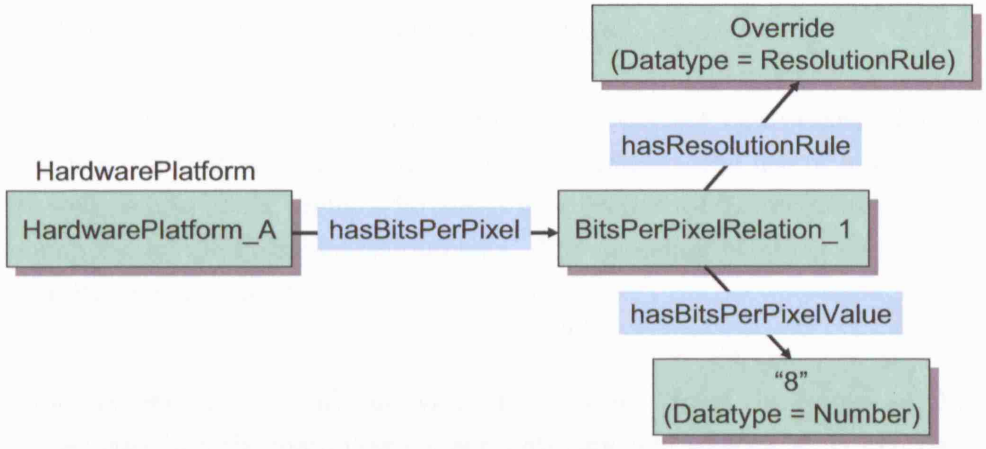


Figure 4.7: Modeling the n-ary relation in OWL

`Override`.

One approach to problems of this type involves the introduction of an artificial individual that explicitly and exclusively binds all the individuals and properties involved in the n-ary relation [153]. Figure 4.7 illustrates this as applied to the UAProf `BitsPerPixel` attribute. An individual called `BitsPerPixelRelation_1` is introduced to stand for the instance of the relation and relate the properties involved in the relation.

To reflect this approach in the ontology, for *each* UAProf attribute, the following were created:

- one OWL class to hold the equivalent relation instances,
- one OWL object property relating the component with this relation instance,
- one OWL datatype property for the value of the attribute, and

- one OWL datatype property for the attribute's resolution rule

to represent the corresponding n-ary relation. For example, for the UAProf attribute `BitsPerPixel`, the following were respectively created:

- an OWL class `BitsPerPixelRelation`,
- an OWL object property `hasBitsPerPixel`, with domain `HardwarePlatform` and range `BitsPerPixelRelation`,
- an OWL datatype property `hasBitsPerPixelValue`, and
- an OWL datatype property `hasResolutionRule`.

Since these new OWL classes and properties are not exact equivalents of the original UAProf attributes, they are named differently; in addition, the use of lowercase starting letters as well as `has` or `is` prefixes for these new properties better matches the naming conventions used in the flow context ontology. The modeling of other UAProf attributes is shown in Figures 4.8 and 4.9.

Part-whole relations To fully integrate the concepts from the imported IPFIX and UAProf ontologies into the main flow context ontology, and to accurately express relationships between concepts that are components or parts of other concepts, it was necessary to be able to express and encode what are called *part-whole relations*. Several concepts in the ontology are actually related by `hasPart` and `isPartOf` relationships and their various specializations. For example, each `L3_IPv4_TwoWaySimpleFlow` logically consists of two `L3_IPv4_OneWaySimpleFlows`; in the ontology they are related by `hasConstituentFlow`. Similar relationships exist, for instance, between an IP packet and its header, between the entire IP packet header and its various fields, or between a network node and its components.

To model this, tripartite concept encoding or modified Structure-Entity-Part (SEP) triplets [154, 155] are used. This is illustrated using the example shown in Figure 4.10, where a flow's `Path` is said to consist of one or more `Links`. One cannot directly subsume `Link` under `Path` because `Link` is not a subconcept of `Path`, that is, the relationship `Link is-a Path` does not hold. However, one may say that `Link` is a `componentOf Path`; alternatively `Link is-a PathComponent`.

The modified SEP triplet approach then suggests the introduction of an artificial concept to express the union of the whole and its parts. In this example, `Path_OR_PathComponent`

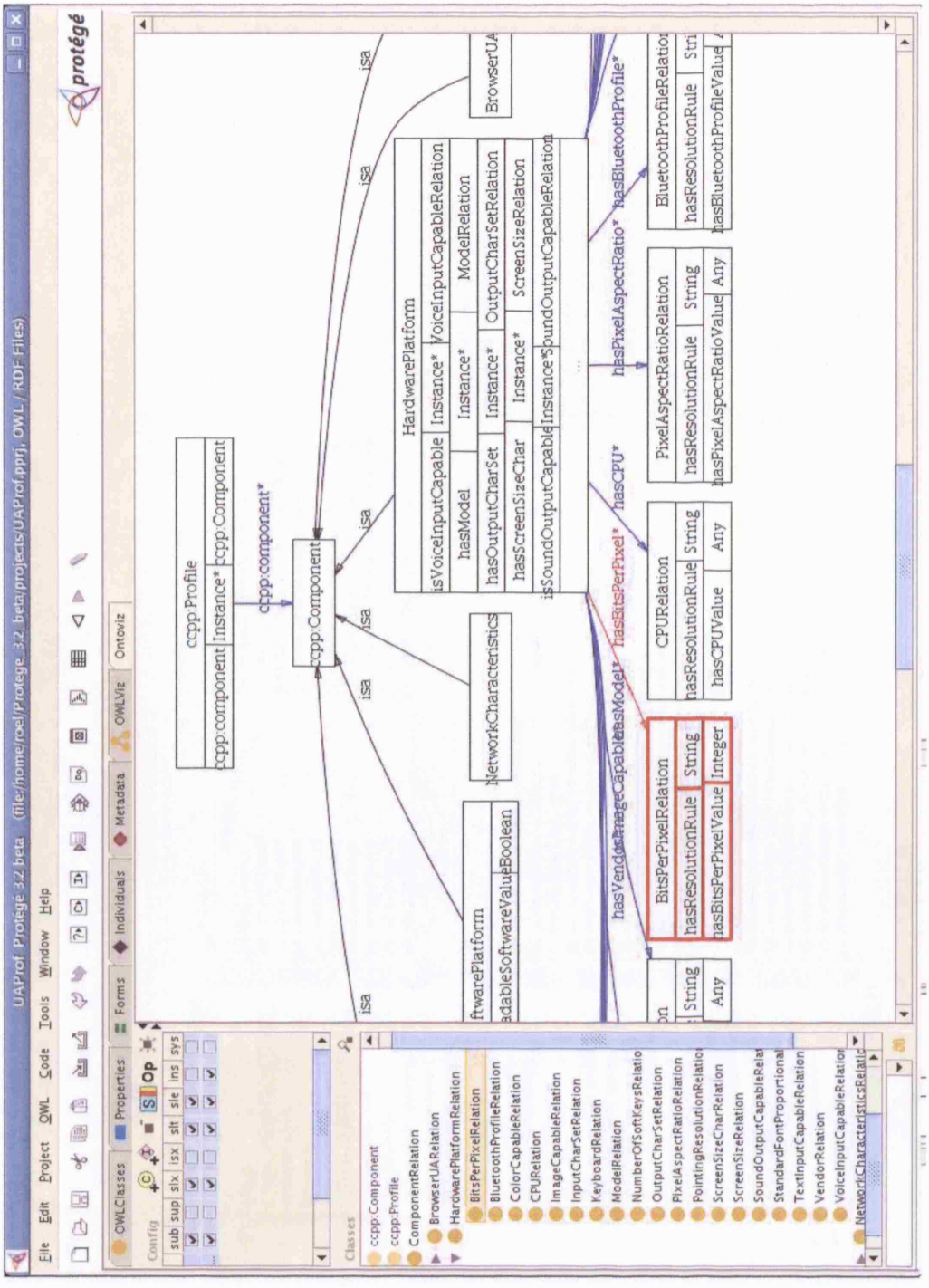


Figure 4.8: UAProf attributes modeled using n-ary relations, as viewed using the Ontoviz ontology visualization application

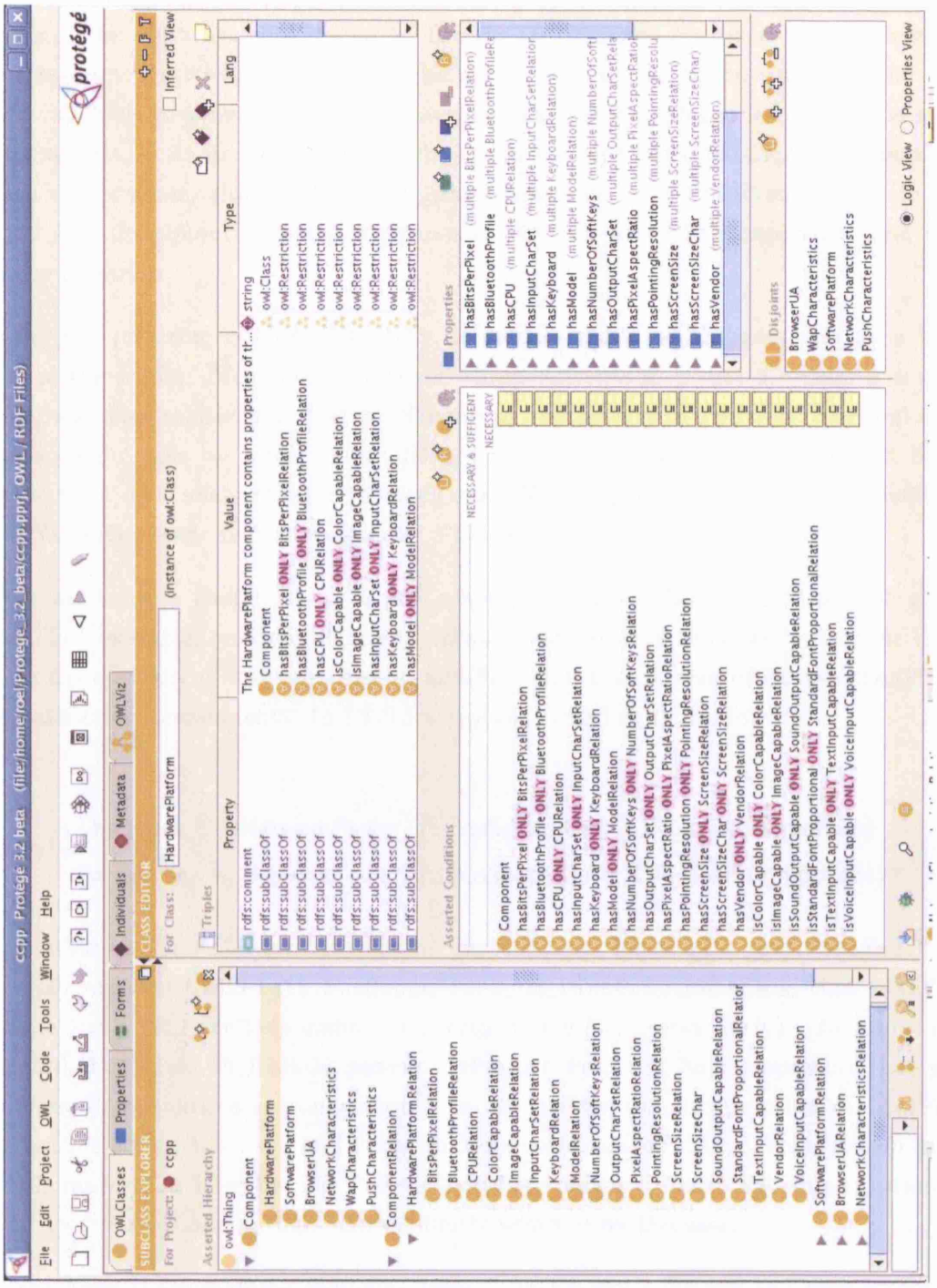


Figure 4.9: UAProf attributes modeled using n-ary relations, showing class definitions in Protégé.

as well as Link_OR_LinkComponent are introduced. Path_OR_PathComponent directly subsumes both Path and PathComponent, because both Path and PathComponent are subconcepts of the union concept – that is, from a set-theoretic perspective, any element of these sets (concepts) would necessarily be an element of the union set (concept). Further down, PathComponent logically subsumes Link_OR_LinkComponent, because a link, its components, or their logical union (the union concept Link_OR_LinkComponent), are all path components, that is, Link_OR_LinkComponent is-a PathComponent. In turn, Link_OR_LinkComponent directly subsumes both Link and LinkComponent, and so on down the hierarchy.

By using the resulting concept hierarchy, inferences made on the part can then be extended to the whole. More formally, given three concepts x, y and z, where y is a part concept, and two relations R and S, S being a subrelation of part-of, the logical implication $xRy \wedge ySz \Rightarrow xRz$ holds [154]. An intuitive example of this would be: *IF* VideoContent isTransportedBy SimpleFlow *AND* SimpleFlow isPartOf FlowBundle, *THEN* VideoContent isTransportedBy FlowBundle (among others).

Another implication that holds in a part-whole relation is $xRy \wedge wRz \wedge ySz \Rightarrow x$ is-a w [154]. For example, one may define LinkFault as a kind of NetworkFault that has, as scope, a Link or any of its components; and PathFault as a form of NetworkFault with scope Path or its components. In DL (description logics) equation form:

$$\mathsf{LinkFault} \equiv \mathsf{NetworkFault} \sqcap (\exists\, \mathsf{hasScope.Link\_OR\_LinkComponent}) \quad (4.1)$$

$$\mathsf{PathFault} \equiv \mathsf{NetworkFault} \sqcap (\exists\, \mathsf{hasScope.Path\_OR\_PathComponent}) \quad (4.2)$$

Although these equations formally define the concepts LinkFault and PathFault, one can simply say that LinkFault hasScope Link_OR_LinkComponent, and that PathFault hasScope Path_OR_PathComponent (i.e., xRy and wRz, respectively). In addition, it was stated that Link_OR_LinkComponent isPartOf Path_OR_PathComponent (i.e. ySz). With these definitions, a reasoner should correctly infer the subsumption of LinkFault under PathFault (x is-a w), and Pathfault under NetworkFault. This is shown in Figure 4.10, marked in blue by the Protégé ontology editor because the subsumption was inferred by the reasoner, rather than explicitly asserted by the user.

After resolving the remaining modeling issues using n-ary relations and part-whole relations, the concepts from both IPFIX and UAProf were integrated into the flow context ontology, and more concise descriptions of the relationships among the concepts in the integrated whole were provided. Figures 4.11 and 4.12 show excerpts from the integrated

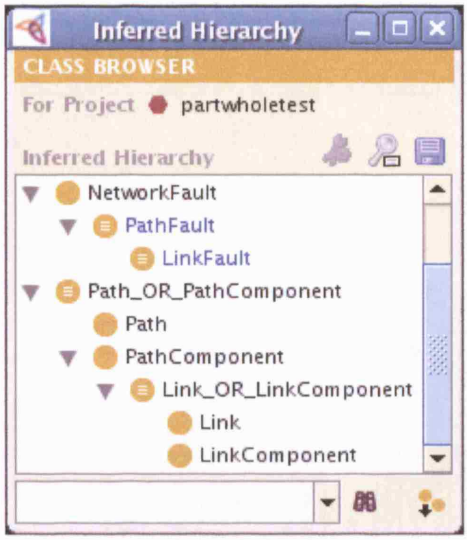


Figure 4.10: Inferred part-whole relations

ontology, visualized using the OntoViz ontology visualization application. These actually only show small fragments of the fully-integrated ontology, as the current version now contains more than 800 classes and more than 100 properties, and could not be displayed in a manner that would show any legible detail.

4.4 Related work

The ontology for flows and flow context presented here is quite novel. Although information models for flows such as IPFIX do exist, these have not been expressed as formal ontologies using a standard ontology language with computational guarantees such as OWL DL. However, flow context as a *concept*, and its corresponding ontology, are both meant to be *integrative* in nature – that is, both are supposed to interrelate and interlink with similar models, conceptualizations, and ontologies, rather than compete with them. Thus, some related work that could potentially be integrated with the flow context ontology, particularly those that deal with network-related concepts, are briefly reviewed in this section.

The Context Ontology (CONON) developed by Wang et al. for pervasive computing applications is structured into two levels, namely an upper-level ontology that captures general concepts about context, and lower domain-specific ontologies tailored to specific application areas [91]. Some of the concepts enumerated in CONON's upper-level ontology

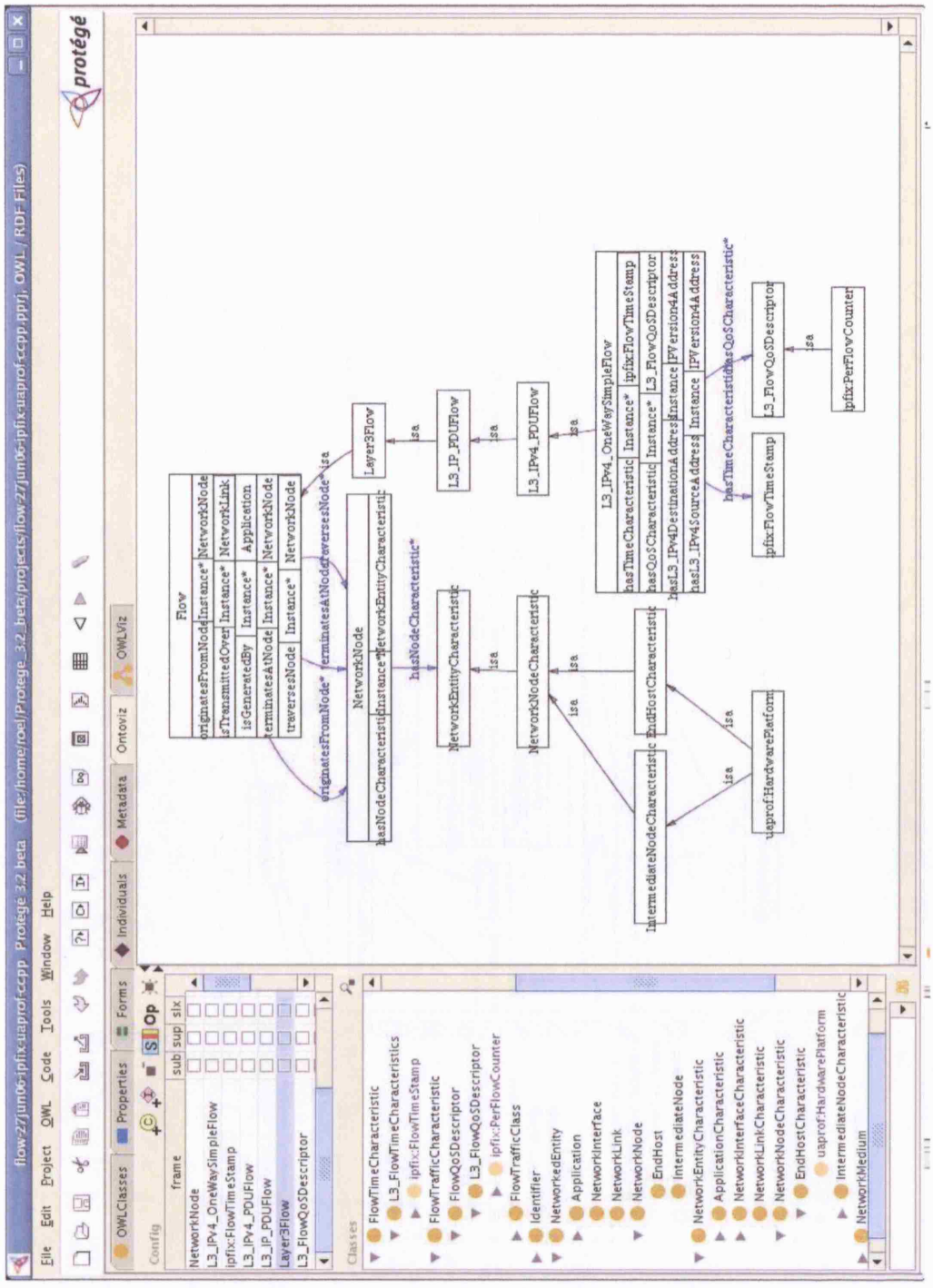


Figure 4.11: Excerpt from ontology, showing integration of IPFIX and UAProf concepts

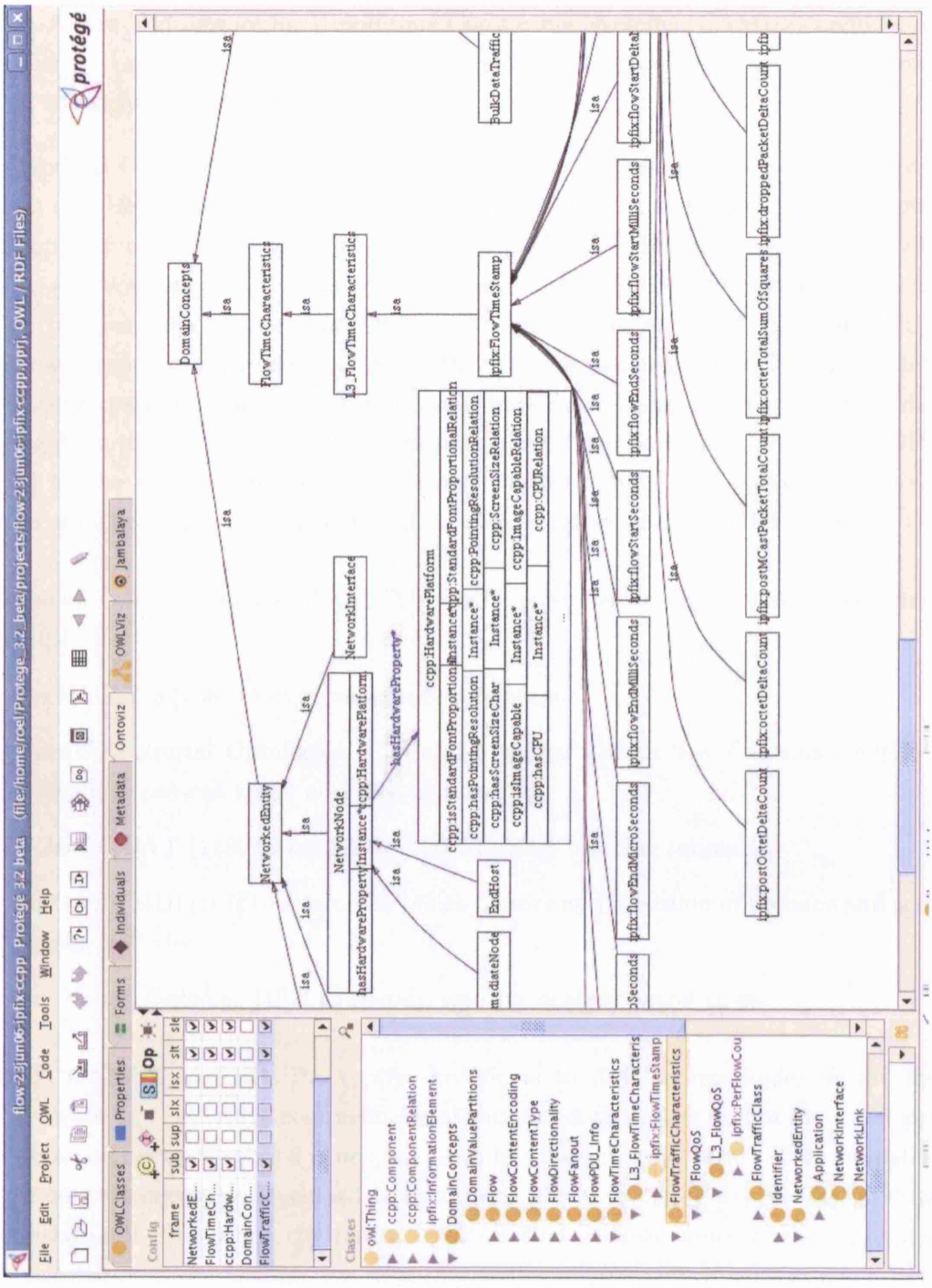


Figure 4.12: Another view showing the integration of IPFIX and UAProf concepts

include `Location`, `Person`, `Activity` and `CompEntity`. Under `CompEntity` one may find the subconcepts `Service`, `Application`, `Device`, `Network` and `Agent`. On the other hand, the Context-Aware Middleware for Ubiquitous Computing Systems (CAMUS) architecture by Shehzad et al. provides the following domain concepts in its ontology: `Agent`, `Environment`, `Device`, `Location`, and `Time` [107].

The Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) developed by Chen and his colleagues at the University of Maryland is designed to model and provide support for pervasive computing applications [156]. This ontology includes vocabularies to express concepts associated with persons, agents, belief-desire-intention (BDI), actions, policies, time, spaces and events. An interesting approach taken in SOUPA is to borrow terms from other ontologies rather than importing them directly in order to limit the overhead associated with wholesale importation and avoid importing concepts that might be irrelevant to the application at hand. The borrowed terms in SOUPA are mapped to the foreign ontology terms using the constructs `owl:equivalentClass` and `owl:equivalentProperty`. The external ontologies referenced by SOUPA include the:

- *Friend-Of-A-Friend Ontology (FOAF)* [157] for personal information and relationships
- *DAML-Time* [158] to express temporal concepts
- *OpenCyC Spatial Ontologies* [159] and *Regional Connection Calculus* [160] to symbolically represent space and spatial relations
- *COBRA-ONT* [118] to model context in smart meeting rooms
- MoGATU BDI [161] to model the beliefs, desire and intention of humans and software agents, and the
- *Rei Policy Ontology* [162] to specify security access control rules.

CONON, CAMUS and SOUPA typify the efforts to develop ontologies in the area of context-aware and pervasive computing applications, in that they do not focus nor provide much detail on network-related concepts, much less flow-based context. This indicates that the flow context ontology developed in this thesis fills a necessary gap in the development of pervasive and ubiquitous computing systems that include context-aware applications and networks.

On the other hand, the work by Laukkanen, Helin and Laamanen [163, 164], Helin and Luakkanen [122], and Helin [165] aim to develop an ontology for wireless networks, particularly to support nomadic applications. In [165], Helin focused on the development of

a QoS ontology for this purpose, defining a vocabulary for QoS that includes concepts such as `Line-rate`, `Throughput`, `Delay`, `Round-Trip Time`, `Bit-error-rate`, `Omission Rate`, `Frame Error Rate`, `Mean Up Time`, `Connection Setup Delay`, `Connection Setup Failure Probability`, `Status`, and `Availability`. The work was further expanded in the EU-IST Ambient Networks (AN) project to include generic network-related concepts such as `Network` and `WirelessNetwork` classes [122, 164], concepts related to AN nodes and networks, and even cost and charging [164, 166]. Although their work did not focus on flows per se, concepts such as these would prove very useful in characterizing flows in terms of QoS metrics other than those defined in IPFIX. The cost ontology on the other hand would be valuable for per-flow billing and charging, or to adapt or influence the routing of flows based on path costs.

Khedr and Karmouch [89, 167], as well as Khedr, Karmouch, Ganna and Horlait [168] presented ontologies for pervasive environments. A `ContextFeature` subsumed classes that describe concepts related to locations, actors, networks, services and actions [89]. These were further developed and expanded in [167] to include network topologies, network profiles, social situations and roles. In [168] they expanded this further to provide support for policies. Their work is also very much similar the ontology described in this thesis and overlaps in several areas, but like the work by Helin et al., fail to provide conceptualizations specific to flows. Nonetheless, the detailed development and "balanced" treatment provided to network-related concepts in their ontology is an important and useful aspect. Other interesting aspects they planned to explore included ontology extensions that would support fuzzy inference constructs to let agents reason about context with a degree of uncertainty [167].

Vergara et. al used ontologies to express various network management models to achieve interoperability and integration of information definitions specified by different management languages such as Structure of Management Information (SMI), Guidelines for the Definition of Managed Objects (GDMO), Managed Object Format (MOF), and others [169, 170, 171]. They argued that this was necessary because existing interoperability approaches such as IIMC (ISO-Internet Management Coexistence), JIDM (Joint Inter-Domain Management), and CIM (Common Information Model) only provide syntactic transformations between models [169]. To translate management specifications, they used a Protégé plug-in to load Internet MIBs and CIM schemas as ontologies [171]. Since integration with ontologies outside the domain of network management was not their primary goal, the flow and flow context ontology can provide the framework for the integration of their work with concepts outside the scope of their domain. For example, an ontological link could be established between their classes that deal with networked devices and the `NetworkNodeCharacteristic` class in the flow context ontology.

A substantial and detailed amount of related ontology work is being undertaken by the Foundation for Intelligent Physical Agents (FIPA),[17] an IEEE Computer Society standards organization that promotes agent-based technology and their interoperability with other technologies. FIPA has published some standards specifying ontologies that software agents can use when communicating about, among others, devices [120] and QoS [121], and to support nomadic applications [172]. The device ontology, for instance, provides descriptions for the properties of devices, product information, hardware, connection information, user interfaces (including further details on display screen properties), memory and software. FIPA's device ontology specification document recognizes that there is an overlap between FIPA's definitions and the work in CC/PP and UAProf, and provides an informative example on how to use the `fipa-device` ontology via CC/PP descriptions [120]. In addition to those specifications that have attained "Standard" status within FIPA, there are also a number of interesting ontologies that can be found in specifications that they currently classify as "Experimental." Some examples of ontologies found in experimental-status specifications deal with diverse agent applications such as in travel-related services, audio/video entertainment and broadcasting, virtual private network (VPN) provisioning, and personal assistants.[18] There is also a standard specification, including an ontology, for ontology services, that is, for services that provide ontology access and related services to agent communities. More recent on-going work by the P2P Nomadic Agents Working Group[19] is aimed at defining a specification for P2P Nomadic Agents, capable of running on small or embedded devices, and to support distributed implementation of applications for consumer devices, cellular communications and robots, and others, over a pure P2P network. However, FIPA has not yet released a specification and ontology dealing with flows, although numerous potential areas for integration exist. For example, `fipa-qos` [121] may be used to describe the QoS of communication channels transporting flows, while `fipa-device` [120] can be used as a vocabulary describing devices that generate, process, forward, or consume flows. Conversely, it would also be beneficial if FIPA-compliant agents could have a standardized vocabulary about flows and the framework to relate this vocabulary with the other FIPA ontologies.

It is possible to envisage flows at various levels of abstraction, and in fact this has been reflected in the flow ontology in this thesis. Therefore, ontologies that can be used to describe "higher-layer" flows such as application-level flows, or perhaps multimedia flows, or other types of flow content, can be reused an linked to the flow ontology in this chapter. For example, the MPEG-7 "Multimedia Content Description Interface" ISO/IEC standard

[17] http://www.fipa.org/

[18] The full set of experimental FIPA specifications, including those mentioned here, may be obtained at http://www.fipa.org/repository/experimentalspecs.php3

[19] http://www.fipa.org/subgroups/P2PNA-WG.html

developed by the Moving Pictures Experts Group allows the description of multimedia content in a form understandable by both humans and machines [173]. In particular, MPEG-7 Multimedia Description Schemes (DSs) provide a standardized way of describing, in XML, concepts related to audio-visual content description and content management for searching, indexing, filtering, and access. These span a wide range of information about the content, including information about its creation, use, its storage features, its spatial and temporal structure, its low-level features, and conceptual information about the reality it captures. For example, the `MediaFormat` descriptor might contain descriptions about the coding format of the media, while `MediaTranscodingHints` might specify transcoding schemes supported by the associated media. Information such as these would enable other types of flow context information to be inferred, such as its QoS characteristics and requirements, or the types of adaptation that may be done in the event of QoS contract violations either by the flow or by the underlying network. Additionally, some work has also been done by Tsinaraki and colleagues at the Technical University of Crete on coupling MPEG-7 and TV-Anytime Forum[20] compliant-descriptions with domain-specific multimedia descriptions written in OWL, and vice-versa [174, 175]. Such work is significant as it points to the possibility of linking the ontology work from MPEG-7 TV-Anytime work with an OWL ontology such as the one developed in this thesis.

4.5 Chapter summary and recommendations

In this chapter a semantic model for flows and flow context was developed by constructing an ontology that defines some of its major concepts, their properties, and their interrelationships. The OWL DL ontology language, a part of the technology recommendations for the Semantic Web, was used to construct the ontology. The implementation of the ontology, including its major classes, the class hierarchy structure, and the properties of these classes, was discussed.

To illustrate and give emphasis to the importance of reusing concepts from external models and ontologies, concepts from at least three existing and ongoing standardization efforts, namely IPFIX, CC/PP and UAProf were integrated into the flow context ontology. Some relatively new approaches suggested in the literature to model special relations such as n-ary relations and part-whole relations were likewise applied. These approaches have not been applied extensively to ontologies related to context-awareness and pervasive computing, nor in context-aware networking.

[20] http://www.tv-anytime.org

The experience with the integration of concepts from IPFIX, CC/PP and UAProf showed that the process can potentially result in loss of information or introduce some artifacts. While creative approaches in the definition of new datatypes, in modeling n-ary relations, and in modeling part-whole relations were creatively applied, these modeling approaches required the definition of artificial classes and relations that otherwise did not exist in the original models. This leads to the following recommendations:

First, ontology languages such as OWL DL should be continuously evolved to improve the richness of their vocabulary, while maintaining maintaining computational guarantees. Although some recommended approaches in modeling special cases such as n-ary relations and part-whole relations were used, these can potentially introduce some unwanted artifacts during the modeling process and sometimes even lead to loss of information. This can lead to difficulties in translating, integrating or interworking different ontologies with each other – an activity which is expected to become more common in pervasive and ubiquitous computing systems and in context-aware networks.

Conversely, ongoing and future standardization work (such as in the Internet Engineering Task Force and in FIPA) should also consider the use of languages such as OWL DL to express models or specifications. Although models such as IPFIX tend to focus on the textual definition, formats and syntax of their information elements (which is perfectly understandable, given the target application), it would be useful for the "semantic future" if the standardization bodies, even on an informative (if not normative) basis, would capture, express and publish the intended semantics using standard (and arguably more popular) ontology languages like OWL-DL, for possible use in other applications. This would help avoid any loss or distortion when porting from the original information model in the specification to another standard language like OWL. It should be stated though, that the mere use of a common language such as OWL-DL would not necessarily guarantee ontology integration: modeling errors such as ontology "misalignment" can still occur [176]. The key to interoperability would still be correct, collaborative, consistent and compatible ontology design and implementation.

Finally, there are system and operational issues associated with the use of ontologies. These are discussed in Section 7.1.4.

Chapter 5

Sensing Flow Context

The previous chapters mentioned that flow context has both intrinsic and extrinsic aspects, and involves information concerning the flow itself and entities that are related to, or even external to it. The next logical question now is: how is this information obtained?

This chapter deals with the issue of *sensing* flow context, which may be considered the first stage in its notional life cycle. Before a context-aware network can use flow context, the information has to be sensed, thus, there should be an infrastructure of *sensors* to do this task.

Context sensors are defined as any software or hardware that can perform context sensing. The term "sensors" is used here in a more general and abstract sense, than, for example the largely hardware-based sensors used in ubiquitous computing applications, or say in process control and instrumentation. Broadly defined, a sensor might be any hardware or software that can "sense" either explicit context or implicit context.

Because there may be a need to develop, integrate or otherwise deploy new context sensing functionalities, it is essential to know how they may be implemented. In the next few sections, three archetypal sensing mechanisms that deal with a few subtypes from the wide spectrum of flow context classes are considered. The design and implementation of sensors for intrinsic flow context, and for two examples of extrinsic context: node and device characteristics, and node location are discussed, and some insights into the requirements for the implementation of similar sensing infrastructures are provided.

5.1 Sensing intrinsic flow context

Flow context may be sensed within the network itself, using existing devices such as network nodes. Routers often keep local, per-flow or per-interface information such as flow rates, flow volumes, and packet loss rates, made available by embedded instrumentation like Cisco System's NetFlow [177] or sFlow [178]. In the Linux operating system, a subset of this information may be derived via the `iptables` facility [144]. However, routers often have a very restricted *global* view of the state of the network, often limited to the network's topological state as inferred from routing exchanges, as well as the bare minimum of information about the flows that traverse them, perhaps partly due to the end-to-end design principle [179].

The SNMP Management Information Base in most routers may also provide a wealth of information about the router itself, including its capabilities and resources, system information, network interfaces, and interface-related statistics [180, 181]. Other network devices, including middleboxes such as network firewalls and intrusion detection devices, may sense other types of flow context, such as the flow's content, and implicitly, the activity or intention of the user generating that flow.

As will be seen in Section 5.4.1, the embedded sensing functionalities in most network devices however either have limited capability, or have limited *dimensionality* [182], logically resulting in limited flow context being made readily available to prospective consumers of the information. As an alternative, an implementation that attempts to combine different flow context sensing functionalities into one package is presented, with the view that the richer information made available can be combined in a synergistic way to infer or uncover more information about the flow. The resulting information would then be potentially more useful to a prospective consumer.

5.1.1 A prototype flow sensor

The prototype application developed in this thesis is called `FlowSensor`, with functional components as shown in Figure 5.1. In implementing this application, an object-oriented approach was adopted, building on the `jpcap`[1] packet capture classes. `FlowSensor` captures IP packets from the wire, demultiplexes them according to their IP protocol type (TCP, UDP, ICMP, IGMP, etc) and passes them on to their respective per-protocol flow table manager objects. Although the packets are classified according to protocol, flow table managers at this point still basically deal with layer-3 (L3) flows, since the protocol data

[1] http://jpcap.sourceforge.net/

units are still full layer-3 PDUs.

Each L3 flow table manager maintains a hash table of active unidirectional flows. The hash key used by each table depends on a tuple that uniquely identifies a basic unidirectional flow in that protocol, as specified by the ontology. For example, for the case of basic unidirectional UDP flows, the hash key is formed from the tuple ⟨source IP address, source UDP port, destination IP address, destination UDP port⟩. The L3 flow table manager checks the incoming packet if it matches any of the existing flows; if it does, it is passed to an L3 flow object that corresponds to that flow. Otherwise, if a match is not found, the manager creates a new layer-3 flow object, creates a corresponding record in the flow table, and passes the packet to the new flow object. This process is implemented by the following code fragment in Java:

```
String hashKey = getHashKey(ipPacket);
L3OneWayIPPacketFlow oneWayFlow = null;

if (!(flowTable.containsKey(hashKey))) {
    //No match found, create new flow object and table entry
    oneWayFlow = createOneWayFlowObject(ipPacket, hashKey);
    flowTable.put(hashKey, oneWayFlow);
}
else {
    //Match found, retrieving existing entry in flow table
    oneWayFlow = (L3OneWayIPPacketFlow) flowTable.get(hashKey);
}
// Dispatch packet to appropriate flow object
oneWayFlow.processPacket(ipPacket);
```

Each L3 flow object senses multiple forms of flow context such as the flow's statistics, traffic characteristics, the protocol or application class of the higher-layer flow that it is transporting, the type of content in its payload, and other information. This encapsulates the philosophy of *multi-dimensionality*: different and sometimes independent aspects of the flow are simultaneously sensed, whenever possible.

Signature scanning

`FlowSensor` senses information about the flow's payload through *signature scanning*, a technique typically used in intrusion detection systems (IDS) to determine if a flow presents a threat [55, 183], or in other systems to determine the application class of the payload

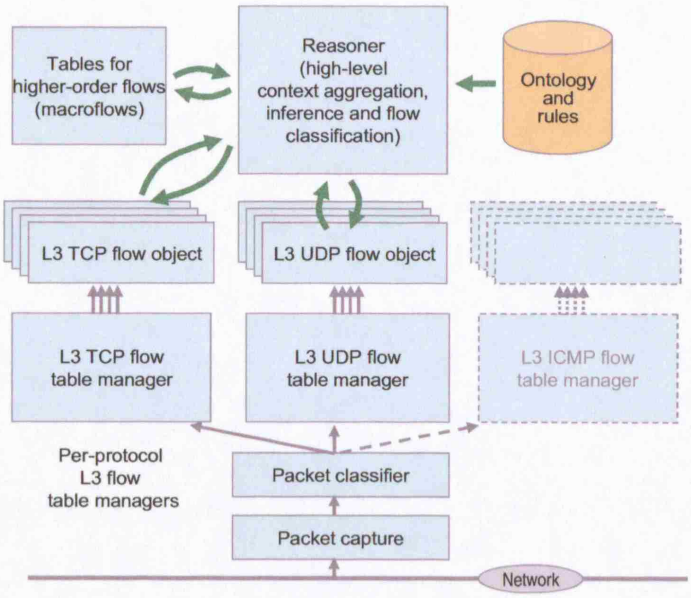


Figure 5.1: `FlowSensor` functional components

PDU (e.g. peer-to-peer, WWW, FTP) or its content type (e.g. audio, video) [184]. The main idea behind signature scanning is to look for patterns or byte sequences – signatures – within the flow that tend to indicate, to a certain degree of accuracy, the nature or properties of the flow.

Signature scanning is provided in `FlowSensor` as a common service to L3 flow objects of the same protocol class by their respective L3 flow table managers. This avoids a costly replication of the set of signatures of interest, which tends to be common across flows of the same protocol, on each flow object.

Signature formats The specifiers for the patterns used in `FlowSensor` to search for signatures are based on formats (filter rules) used in other existing experimental and deployed applications. The argument for supporting existing formats is that there is bound to be a large and substantial set of signatures developed for each application, that are already available and in use in the community. Unless there are compelling reasons to invent a new format, it would be advantageous just to reuse these existing ones. In addition, while the existing filter specifications by themselves are quite flexible and powerful, one main advantage presented in this work is that it is explicitly intended to interoperate with *multiple* formats, enabling a larger base of signatures to be reused. Some of these existing filter formats are now briefly reviewed:

- *BLINC* ("blind classification") is from the work by Karagiannis, Papagiannaki and Faloutsos on multilevel traffic classification [184]. Signatures in BLINC are specified as comma-separated byte sequences, either in hexademical (delimited by '\') or as character strings enclosed in quotation marks. Numbers in parenthesis, if present, denote the starting byte in the payload where the sequence is found; otherwise the sequence is assumed to be found at the start of the payload. The keyword `plen`, if present, denotes the size of the payload. Operators for AND ('&&') and OR ('||') are supported. For example, the BLINC signature specification:

  ```
  \x47\x4e\x44, \x00\x01\x00\x00\x00\x00\x00(16,plen=23), "LIME"(23)
  ```

  contains patterns that might indicate the payload of the Gnutella[2] peer-to-peer protocol.

- *u32* is a filter available for IP packet classification and filtering in the Linux operating system [144]. A `u32` *selector* specifies bit patterns that can be used to match with a packet being processed. The general selector syntax is as follows:

  ```
  match [u32|u16|u8] PATTERN MASK [at OFFSET|nexthdr+OFFSET]
  ```

  where `PATTERN` and `MASK` together specify a bit pattern to be matched, `OFFSET` specifies an offset from the start of an IP header, `nexthdr+OFFSET` specifies an offset from the start of an upper-layer protocol header, and `u32`, `u16` or `u8` specify the size of the mask in bits. For example, the `u32` selector

  ```
  match u8 64 0xff at 8
  ```

  matches IP packets with a time-to-live (TTL) value of 64, since the TTL field is the 8th byte in the IP header.

- *Snort* [55] is an open-source network intrusion detection system originally written by Martin Roesch. It is basically a packet sniffer coupled to a real-time packet decoder and a detection engine that performs content matching and protocol analysis to detect a variety of attacks and probes. The detection engine is programmed using rules that describe per-packet tests and actions. Although a full discussion on rule construction in Snort is beyond the scope of this document[3], some of the essential features are described below.

  Snort rules consist of two logical parts: the *header* and *options*. The header is structured as follows:

[2] http://www.the-gdf.org/

[3] The interested reader may refer to [185] for a tutorial on constructing Snort rules

```
action protocol address port direction address port
```

where

- `action` specifies what actions should be taken when the rule is met, such as to pass the packet, log it, send an alert message, activate another rule, and other actions,
- `protocol` specifies whether the rule should apply to IP, TCP, UDP or ICMP packets,
- `addresses` specify source and destination IP addresses or address ranges, or to `any` address, and
- `ports` specify port numbers or port number ranges.

The option part of the rule is enclosed in parentheses and consists of one or more keywords and any applicable arguments. Multiple options are separated by semicolons, forming a logical AND between them. Among the keywords recognized by Snort, one interesting example is the `content` keyword which allows a pattern to be searched inside a packet. For example, the following Snort rule[4]

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any
(msg:"COMMUNITY INAPPROPRIATE girls gone wild"; content:"girls";
nocase; content:"gone"; nocase; content:"wild"; nocase;
flow:to_client,established; classtype:kickass-porn; sid:100000124;
rev:1;)
```

issues an alert when a packet with a TCP payload and with a TCP port number corresponding to the HTTP well-known port, originating from the external network, going into any port on any host within the local (home) network, on an active TCP connection, contains (among others) the case-insensitive strings "girls" and "gone" and "wild."

- *SpamAssassin*[5] is an open-source mail filter used to identify spam, or unsolicited commercial email messages. SpamAssassin supports a variety of mechanisms including header and text analysis using hand-coded rules, statistical (Bayesian) filtering and learning, blocklists based on domain names, and collaborative filtering databases.

  SpamAssassin rules have three parts, including (1) a type and regex, (2) the score, and (3) the description. The type and regex part specifies a regular expression (regex)

[4]From http://www.snort.org/pub-bin/downloads.cgi/Download/comm_rules/Community-Rules-2.4.tar.gz

[5]http://spamassassin.apache.org/

[186] that can be used to match a pattern within a specific portion or segment of the email specified by the type. The score represents a value given for a match to that rule, while the description contains some notation about the rule. As an example, the following SpamAssassin rule fragment

```
body _DRUGS_ERECTILE2  /\bV(?:agira|igara|iaggra|iaegra)\b/i
```

attempts to search for exact, misspelled, or deliberately obfuscated representations of "Viagra" within the body of an email.

As previously mentioned, the idea is to support a number of these representative formats in order to leverage any existing patterns (filters) developed or used by its user community. At start-up, `FlowSensor` can be instructed to read in a file containing signatures specified in a number of formats. Each entry in this file contains the following fields:

```
targetClass  exprFormat  expr  backReference
```

where

- `targetClass` represents the class(es) and their subclasses to which this search expression applies, that is, flow objects of this class are supposed to scan the flows they monitor for occurrences of this signature,
- `exprFormat` represents the format used by the signature's filter expression, such as BLINC, u32, and others,
- `expr` contains the actual filter expression for the signature
- `backReference` is a string that can be used for comments, or for a message that can be sent to a human operator, or to another program. In the current version, backReference contains an *assertion* that is sent to the reasoner component of `FlowSensor`. (This will be discussed in more detail in Chapter 6.)

`FlowSensor` currently recognizes the following format types: `blinc`, `u32`, `snort`, and `sa` (SpamAssassin). However, only the `blinc` and `u32` modules have been fully implemented in the current version.

Both `blinc` and `u32` only require exact matching between a pattern and a substring of the PDU being searched. In cases where the signature is known to occur at a particular position

within the PDU, a simple sequential byte-by-byte comparison starting at that position is sufficient. When a signature needs to be searched throughout the entire payload, the Boyer-Moore-Horspool (BMH) exact string matching algorithm [187] is used. A discussion of this algorithm, including some of the code used to implement it, is presented in Appendix B.

5.1.2 A visualization tool

The object-oriented approach in implementing `FlowSensor` probably trades off some degree of performance in favor of modularity, ease of design, and implementation speed. This approach has also made it possible in this work to quickly assemble and integrate a visualization tool. This tool is useful as it allows the state of the flow table managers, flow objects, and signature scanning engines, shown in Figure 5.2 to be monitored. It may be of potential use in flow-based network management and other applications, as will be explained in Chapter 7.

5.2 Sensing node and device characteristics

Context sensing functionality can also be performed at end-hosts, where there is rich context information such as the characteristics and capabilities of end-devices, the state and nature of the applications generating the traffic, the content-type and characteristics of the application flows themselves, and the identities and activities of users such as their movement and location. Some forms of context may be obtained using existing host applications, services or resources; a simple example might be the `ps` and `finger` commands in Unix providing some information about process status and user activity and location, respectively.

If some necessary context sensing capabilities are not present, they may be added-on, as experimentally illustrated for some wearable computers [14] and mobile phones [37]. If the context sensing functionality cannot be directly integrated into the end-host or network device, then there may be a need to deploy dedicated context sensing devices and infrastructures. This however assumes that all types of context sensing functionality can be known in advance and either integrated or pre-deployed. Considering the heterogeneity of today's networks and their rapid technological evolution, having fixed and pre-defined context sensing functionality might not always be adequate to the task.

This section discusses the implementation of a sensor that obtains the characteristics of

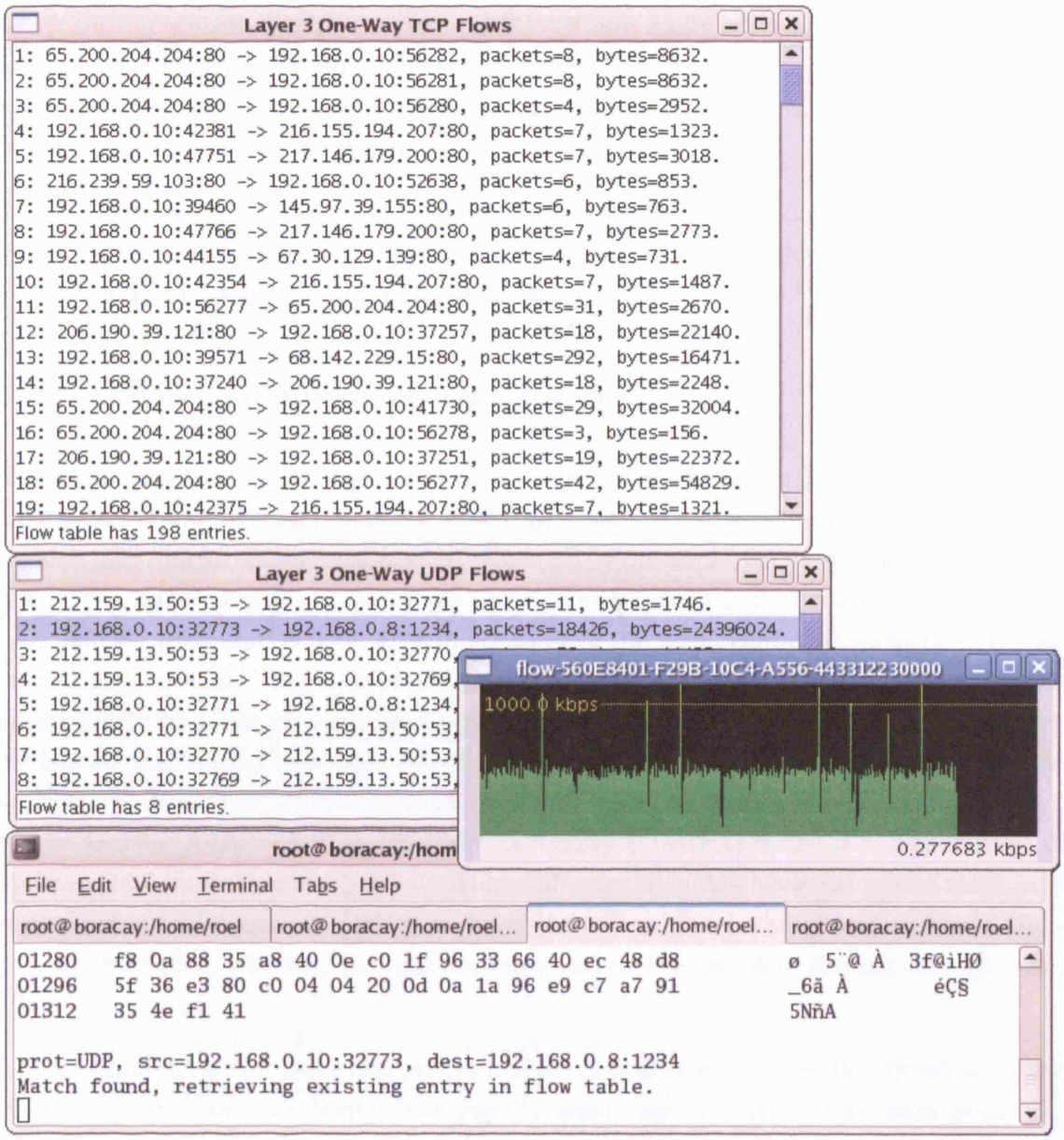


Figure 5.2: A flow visualization tool based on `FlowSensor`. The top two windows show the entries in the TCP and UDP flow tables. A flow entry in the UDP flow table is selected (shown highlighted), opening a window showing its traffic graph. The bottom window shows the status of signature scanning on this flow.

network nodes and end-devices. As will be recalled, in Chapter 3 these types of information were classified under extrinsic flow context, and in Chapter 4 these were modeled and included in the flow context ontology. While the sensor presented uses a simple mechanism to obtain the context information it provides – it essentially relies on the Simple Network Management Protocol (SNMP) [39] – the novelty introduced here is the ability to dynamically deploy this sensor along the current or future path of a flow, using the active and programmable networks paradigm.

The next section provides a very brief overview of active and programmable networks. A specific active network platform called DINA is then discussed, followed by a presentation on the design and implementation of the sensor itself.

5.2.1 Active and programmable networks

The term "programmable networks" refers to a broad class of networks characterized by the availability of open programmable network interfaces, a visible separation between transmission and control software, and a virtualization of the underlying network infrastructure [188]. The main novelty offered by programmable networks is that the network elements may provide *computational* services in addition to the usual communication (e.g. packet forwarding) services. The mode of programmability and degree of "openness" of such nodes varies widely from one architecture to the next, ranging from having a set of open network APIs that can be used by third-party developers, to the ability to inject and execute code while the node is "live." *Active networks*, whose origins may be traced to a program funded by the Defense Advanced Research Projects Agency (DARPA) of the U.S. Department of Defense [189], may be considered a subclass of programmable networks where users may inject customized programs into the nodes of the network, and where these nodes, in addition to forwarding, may also perform computations on the user data flowing through them [190].

The general interest in active and programmable networks stems from the potential ability of these architectures to flexibly and rapidly create and deploy new network protocols, services, overlays and even new network architectures [188]. While pre-deployed overlay networks provide the ability to build new services over existing network infrastructures, network programmability in turn offers the ability to *dynamically* build these new overlays on demand. Services may be flexibly composed from basic component modules in response to the specific flow requirements as inferred from their context: for example, a compressible flow with privacy requirements (e.g. a voice conversation) traversing a low-bandwidth, wireless link may require the combination of encryption and compression

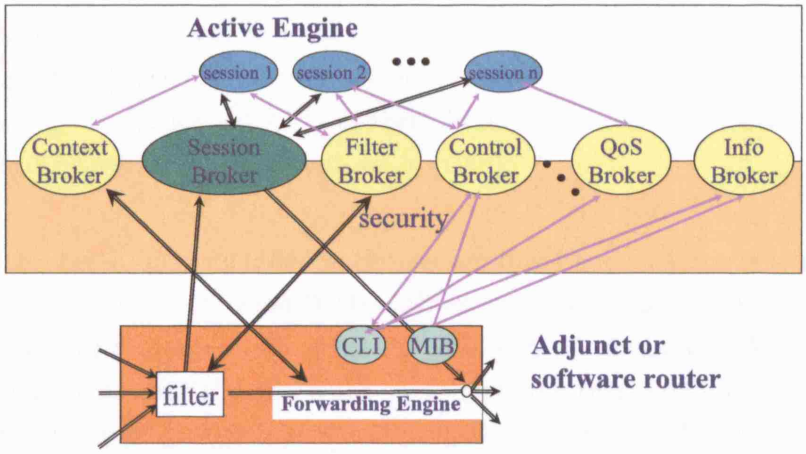


Figure 5.3: Architecture of the DINA active platform [195]

adaptation modules; when the same flow traverses another wireless link with higher bandwidth capacity then only the encryption module will be deployed and executed. In this case, programmability provides the ability to dynamically deploy these service components at relevant points within the network on demand. For example, some application overlays that have been demonstrated using the FunnelWeb active networking platform include content caching, transcoding, multi-metric routing and multicast overlays [191], fault signalling and congestion control overlays [192, 193], and peer-to-peer traffic management overlays [194].

5.2.2 The DINA platform

DINA[6] [195] is an active networking platform developed in the EU IST CONTEXT[7] project, based on concepts and ideas from the Active Bell Labs Engine (ABLE) [196]. A DINA node (Figure 5.3) typically consists of two logically distinct components, namely a Forwarding Element such as a router, and an Active Engine. DINA active packets are encapsulated in UDP with an ANEP header [197], and carry active code payloads written in Java. These active packets are intercepted by a *diverter* and sent to the Active Engine for execution.

[6]The choice of DINA for prototyping and proof-of-concept work in this thesis was primarily a matter of convenience for the author, given his previous work that uses the platform. As the platform's role was merely to demonstrate functionality rather than to achieve the most optimal solution, it was decided that DINA would be sufficient for this purpose, rather than the numerous other platforms available elsewhere.

[7]http://context.upc.es/

DINA employs a number of software modules called *Brokers* which are used to provide active packets restricted access to local node resources and to perform node-local operations. Active packets invoke the services of these brokers by instantiating *Broker Interface* objects and using the predefined APIs of these broker interfaces. Some of its brokers are briefly described below (and fully described in [195]):

- *Session Broker* A filter installed in the Forwarding Element intercepts active packets and sends them to the Session Broker, which in turn forwards them for execution in specified active *sessions* running within Java Virtual Machines (JVM). The Session Broker maintains a database of sessions, keeping information on their state (pending or alive), age, sequence numbers and IDs, IP addresses and UDP ports used, the JVMs used, and security and authorization information. Aging information in the database allows the Session Broker to terminate sessions that are old and unused.

- *Information Broker* The Information Broker provides access to node-related information from the SNMP MIB [198] of the active host. Although it has some built-in primitives for obtaining specific information such as the router's name, the number of its interfaces, the status of these interfaces (up, down), their IP addresses, and others, it has a more general primitive that provides access one or more MIB objects simply by supplying their object IDs (OID) [198].

- *Control Broker* The Control Broker enables active services to configure routing tables and install either temporary or permanent routes in the Forwarding Element. It is also used to configure virtual private network (VPN) connections.

- *Network Broker* The Network Broker provides sessions with the capability to establish communication services such as sending out UDP datagrams, establishing TCP connections, or setting up server sockets.

- *Filter Broker* The Filter Broker provides a means to control the underlying (platform-dependent) packet filtering mechanisms of the router. For example, for a Linux-based active router, Filter Broker provides an interface to control the rules in `iptables` [144].

- *QoS Broker* The QoS Broker provides a facility to configure and manage an active router's support for QoS functionality through the DiffServ architecture [199]. For example, it provides methods to set up DiffServ classifiers, policers, markers, and other components.

- *SIP Broker* The SIP Broker provides access to a hosted Session Initiation Protocol (SIP) [90] application, allowing SIP control and management of SIP entities such as

SIP softswitchies, proxies, and user agents.

- *WLAN Broker* The WLAN broker provides the ability to control and manage attached wireless LAN access points and configure associated wireless networks.

5.2.3 Sensor implementation

To implement the sensor, a Java class that extended the `activeCode` class included in the DINA package was written. `activeCode` essentially instantiates a Session Broker Interface object and causes a session to be created in the Session Broker for the sensor.

The design and functionality of the sensor itself are actually very simple. The basic idea is to inject the sensor as an active packet, and once it is up and running within the active node, for it to (1) install itself as a persistent service, (2) propagate a copy or copies of itself over the network, and (3) listen for and service requests for context information. The main implementing functions are shown in the following (quite intuitive and readable) Java code fragment:

```
public void run()
{
    if(!refreshThreadLaunched) {
        //Execute this block only if we're not
        //the refresher thread
        createBrokers();
        spawnRefresherThread();
        sendSensor();
        createServerSocket();
        processRequests();
    }
    else {
        doRefreshes();
    }
}
```

These method calls are further elaborated below:

- `createBrokers()` - when the sensor is received, the first thing it does is to instantiate broker objects for the Information Broker and the Network Broker. The Information Broker interface will provide facilities to access the underlying SNMP MIB. (The role of the Network Broker interface will be discussed in the other method calls.)

- `spawnRefresherThread()` - as previously mentioned, active sessions are aged and subjected to timeout if they are idle. This method spawns off a thread whose only job is to execute its `doRefreshes()` method. In a refresher thread, `doRefreshes()` loops continuously, executing a `Thread.sleep()` most of the time and periodically "waking up" to invoke the `refresh()` method of its `sessionBrokerInterface` superclass.

- `sendSensor()` - this method sends out a copy of the sensor out to the network, where it will be intercepted and executed if it encounters another DINA node. A number of models for propagating the sensor are possible: (a) it can be propagated along the path of a flow, (b) it can be propagated to a specified next host (not necessarily along a flow path, or (c) it can be broadcast in a directed and structured way to neighboring hosts using the appropriate interfaces.

- `createServerSocket()` - this method uses the Network Broker interface[8] to create a UDP socket that waits for requests from clients and control commands from a management entity.

- `processRequests()` - this method waits for messages on the server socket created by `createServerSocket()`, parses them, and processes them accordingly. It currently supports four main commands by invoking the following methods as appropriate:

  - `getContext(InetAddress addr, int port, String funcArgs)`, which further parses `funcArgs` to obtain an SNMP MIB OID to be passed on to the Information Broker interface object. The returned value is sent back to the requesting client at IP address `addr` on UDP port `port`.

  - `subscribeRequest(InetAddress addr, int port, String funcArgs)`, which allows clients to request to be subscribed or unsubscribed to continuous streams of context values. For example, a client might want to regularly receive information on the location of the node, the amount of traffic on an interface, or perhaps the processor load.

  - `notifyRequest(InetAddress addr, int port, String funcArgs)`, which allows clients to request to be notified in response to changes in context values. For example, a client may wish to be notified if an additional network interface changes in status from "down" to "up," possibly to evaluate an option to re-route a flow.

  - `killSensor()` terminates the operation of the sensor.

[8] Although the author was not involved in the development of the DINA platform, he had to make some modifications to DINA's Network Broker interface code during the development of this sensor.

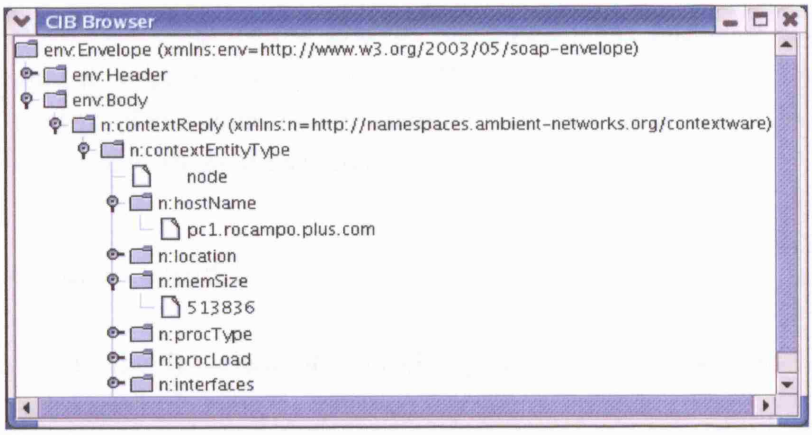


Figure 5.4: The output of a node context sensor

In the first three commands, an argument in `funcArgs` additionally specifies if the returned information should be serialized in XML (preferred) or in a predetermined text format. Figure 5.4 shows a view from a client application that was written[9] to test the functionality of this sensor class.

Node context sensors can either persist indefinitely, or be manually terminated by sending them a *Kill* message. A useful compromise between these two modes would be to specify their lifespan upon creation and to provide support for extensions of these lifespan (similar to refreshes in the Session Broker). These however are not implemented in the current version of the prototype.

5.3 Sensing location

In the previous section a technique for constructing dynamically-deployable sensors for SNMP-derived node-related extrinsic flow context was discussed. One of the objects defined in the SNMP MIB, `sysLocation`, in fact holds information about the location of the node [198]. Although this is more likely intended to describe the relatively static location of a network node such as a router, it could conceivably be updated dynamically to reflect the location of a mobile host as well (using the Net-SNMP[10] command `snmpset`, for instance).

[9]The development of this sensor and the client application shown here preceded the full development of the flow context ontology, so the naming and structure of the context information shown in Figure 5.4 does not necessarily conform to the model in Chapter 4

[10]http://net-snmp.sourceforge.net

This section focuses on how a node's location – a type of extrinsic flow context – may be sensed. A novel technique for fine-grained location sensing based on acoustic spread-spectrum techniques is presented. Although the system described here was developed primarily for a controlled, experimental environment, the design principles and results may be further extended to create versions for real-world deployment.

5.3.1 Flow context and location

Before proceeding to the discussion of the technique, it would be useful to try and answer the question: Is location information relevant?

Location information has played an important role in the emergence of context-aware computing itself, with many experimental efforts centered on the use of user location to trigger or offer services [200]. This in turn has spurred research focusing on the development of location techniques primarily for context-aware computing and for location-based services.

In mobile and ad-hoc networks, location information is used by geographic routing protocols such as GeoCast [201], Distance Routing Effect Algorithm for Mobility (DREAM) [202], Location-Aided Routing (LAR) [203] and Greedy Perimeter Stateless Routing (GPSR) [204]. For instance, GPSR tags each packet with a 12-byte position identifier.

In Section 7.2 an example illustrating the application of location information in aiding the handoff process in a mobile network through flow routing and adaptation is presented, emphasizing the usefulness of having a means to sense location information, similar to what is discussed in this section.

5.3.2 An experimental location sensor

This section discusses the implementation of an experimental location sensing scheme that would be compatible with commercially-available off-the-shelf devices such as a personal digital assistant (PDA). To roughly estimate the position of users within a relatively large area, such as within a 3–4 meter radius, techniques such RADAR [205] and its variants may be used with a PDA outfitted with an IEEE 802.11 interface. For fine-grained positioning, such as within 10–20 cm., while a number of systems such as the Active Bat Location System [206] and Cricket [207] have been discussed in the literature, most of these systems usually use ultrasonic transducers and a channel for transmitter-receiver synchronization such as an RF or infrared channel. However, it would be advantageous to have a position sensing scheme that would use available interfaces and require only a bare minimum of

hardware interfacing, if any at all. An assumption is made here that the located object would operate asynchronously with respect to the positioning system, thus, no separate synchronization channel should be required between the positioning system and the node whose position is being sensed.

Hyperbolic multilateration

A location estimation technique known as hyperbolic multilateration [208] does not require the tracked object to be synchronized with the positioning system. In hyperbolic multilateration, time-synchronized signals are emitted from pairs of signal sources (called *beacons*) and are detected by a receiver. Assuming that signals simultaneously emitted from a pair of beacons i and j arrive at a receiver times t_i and t_j respectively, referenced to the receiver's clock, then the *time-difference-of-arrival* (TDOA) is defined as $t_i - t_j$. Since all of these time measurements are referenced to the receiver's own clock, no synchronization between the beacons and the receiver are necessary. The mathematical details of this technique are provided for the interested reader in Appendix C.

Obtaining TDOAs using spread-spectrum techniques

To obtain time-of-arrivals, and consequently time-difference-of-arrivals, direct sequence spread spectrum (DSSS) [209] modulated acoustic signals were used in this work as beacon signals. In spread spectrum modulation, a digital pseudorandom sequence called a PN (pseudonoise) code is used as a modulation waveform. Each pulse in the PN code is called a *chip*, and the inverse of their time period is called the chip rate. Since the chip rate is typically much higher than the bit rate of the original information, the signal energy gets distributed over a bandwidth much greater than the original information bandwidth, effectively "spreading" its spectrum.

The PN codes used in spread spectrum modulation have a number of interesting properties, particularly in terms of cross-correlation and autocorrelation. Cross-correlation in signal processing refers to a measure of similarity between two signals. In the case of two complex-valued discrete-time sequences f and g, it is defined as a function of the relative time m between them and is given by

$$\phi_{fg}[m] = \sum_{n=-\infty}^{\infty} f[n]g^*[n+m] \tag{5.1}$$

where the asterisk indicates the complex conjugate. The function $\phi_{fg}[m]$ has a maximum value when the two sequences are aligned in such a way that they are most similar to each other. Autocorrelation on the other hand simply refers to the process of cross-correlating a signal with itself; it can be considered a measure of the similarity of a signal with time-shifted versions of itself. The PN codes used in spread spectrum exhibit very sharp autocorrelation functions, that is, the autocorrelation function of a single code sharply peaks at m=0 and has a small value elsewhere. This means that if one correlates an incoming signal modulated with a known PN code with a local copy of the same signal, the correlation function would give a sharp peak when the two signals are perfectly aligned (synchronized) in time. This ability to synchronize an incoming (known) spread-spectrum signal with a local reference copy works reasonably well even in the presence of either narrowband or wideband noise [209], as well as environmental scattering [210]. These properties give spread-spectrum signals a distinct advantage over narrowband signals when used for synchronization.

Since spread-spectrum signals can be used for synchronization, it follows that they can be used to detect when a known signal emanating from a source impinges on a receiver, that is, they can be used to determine a signal's time-of-arrival. However, hyperbolic multilateration also requires that the receiver be able to distinguish the arrivals from two different sources i and j in order to compute the TDOA. To achieve this, i and j should use *different* reference signals whose cross-correlation values are typically much lower than their autocorrelation values. The PN codes used for spread-spectrum modulation are designed to exhibit good cross-correlation properties, that is, codes are selected such that they have very sharp autocorrelation peaks and nominally low (and bounded) cross-correlation values between them.

To generate spread-spectrum beacon signals, Gold codes were used in this thesis as PN codes because of their excellent autocorrelation and cross-correlation properties [211, 212]. The Gold codes used had a sequence length of 127 bits, a chip rate of 10 Kchips/s, and were used in a binary phase-shift keying (BPSK) modulation scheme. Figure 5.5 shows two such beacon signals, generated from distinct Gold codes, impinging on a receiver (an ordinary PC microphone) after being emitted from two PC speakers acting as beacons. The unprocessed composite signal received by the microphone is shown by the waveform on top. The bottom graph shows the result of cross-correlating the composite signal individually with reference copies of the signals from beacon 1 (blue) and beacon 2 (green), plotted on the same time axis. The sharp correlation peaks in the bottom waveform indicate the instances when each beacon arrived at the microphone, and the TDOA is consequently computed as the time distance between these correlation peaks.

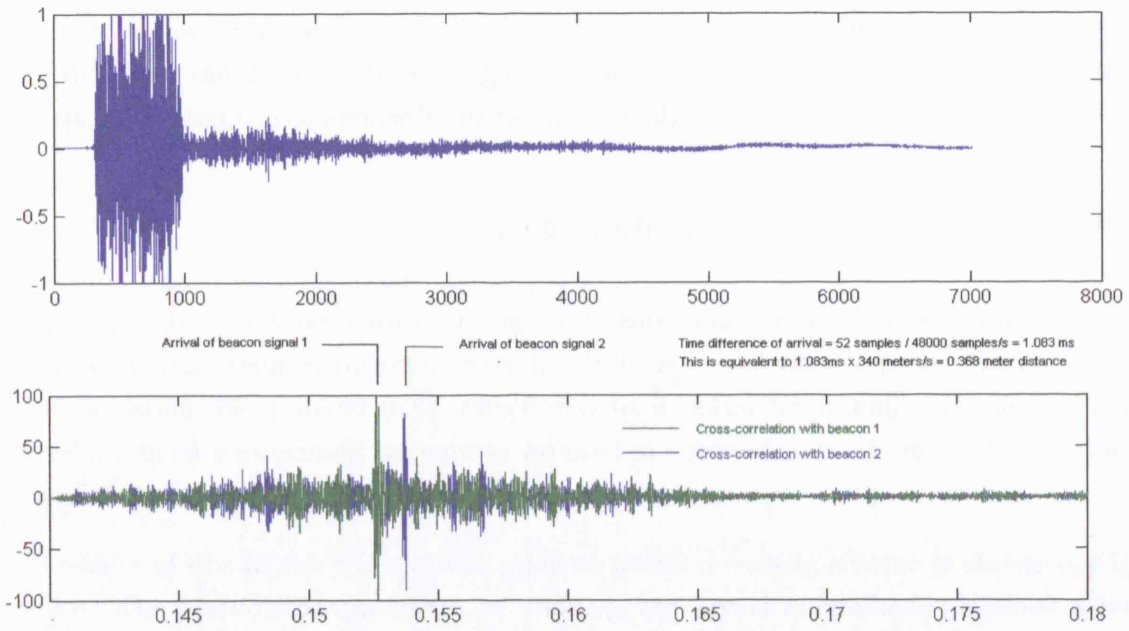


Figure 5.5: Two spread spectrum signals impinging on a receiver. The x-axis of the top graph denotes the index of the audio sample, while its y-axis is its normalized magnitude. The x-axis of the bottom graph denotes relative time in seconds, while the y-axis is the magnitude of the correlation function $\phi_{fg}[m]$.

Results

The experimental set up for this work consisted of four PC speakers, each with a single 3.5-inch driver, positioned in a room. Three were mounted on the ceiling, approximately 2.2m above the floor on average, while one was mounted on a wall, approximately 80.5 cm above the floor level. Acoustic beacon signals, consisting of 127-bit Gold codes with a chip rate of 10 kchips/s and BPSK-modulated with a 10 kHz sine wave, were simultaneously transmitted through the speakers. A microphone (simulating a PDA in the test scenario) recorded the received signal every 10 cm on a 130 cm x 110 cm grid. The recorded signal was then successively correlated with each of the transmitted Gold codes. A correlation peak indicated the instant a beacon signal arrives at the microphone. The speed of sound was approximated to the first order using the formula

$$c \approx 331.5 + 0.610 t_{\text{air}} \tag{5.2}$$

where t_{air} is the air temperature in degrees Celsius and c is in meters/second. In the experiments, the temperature was recorded from a digital thermometer. In an actual implementation, the ambient temperature may be supplied by an online sensor.[11] Alternatively, a fixed approximate value may be used in environments where the temperature is regulated or typically does not vary to a large degree.

The results of one of the trials of the acoustic position sensing scheme is shown in Figure 5.6. The positions marked with "x" indicate the actual microphone positions, while positions marked with "o" indicate the position estimates computed through hyperbolic multilateration. Lines interconnect pairs of actual and computed positions. The gaps in the grid where there are no "x" marks represent points where the computations did not converge within the maximum number of iterations, or the resulting computed position was outside the coordinate system. For the data shown in the figure, the computed position deviated from the actual position by 7.0 cm on average, and 80% of all computed positions deviated by less than 9.4 cm from their actual positions. In sensing the location of people and objects, one is normally more interested in their (x, y) position rather than their elevation above the floor. For the data represented in Figure 5.6, the deviation from the actual positions along the xy–plane was around 4.6 cm on average, and less than 7.5 cm for 90% of all computed positions. The cumulative distribution of errors along the

[11] Gerald Maguire of KTH, in a private communication, also suggested that the propagation speed may be derived from time measurements using a receiver with *a priori* known position. The author later derived the formula $c = (r_i - r_k)/(t_i - t_k)$, which only requires position and TDOA information, which in turn could be obtained using the existing acoustic beacons and a fixed receiver – eliminating the need for a conventional temperature sensor.

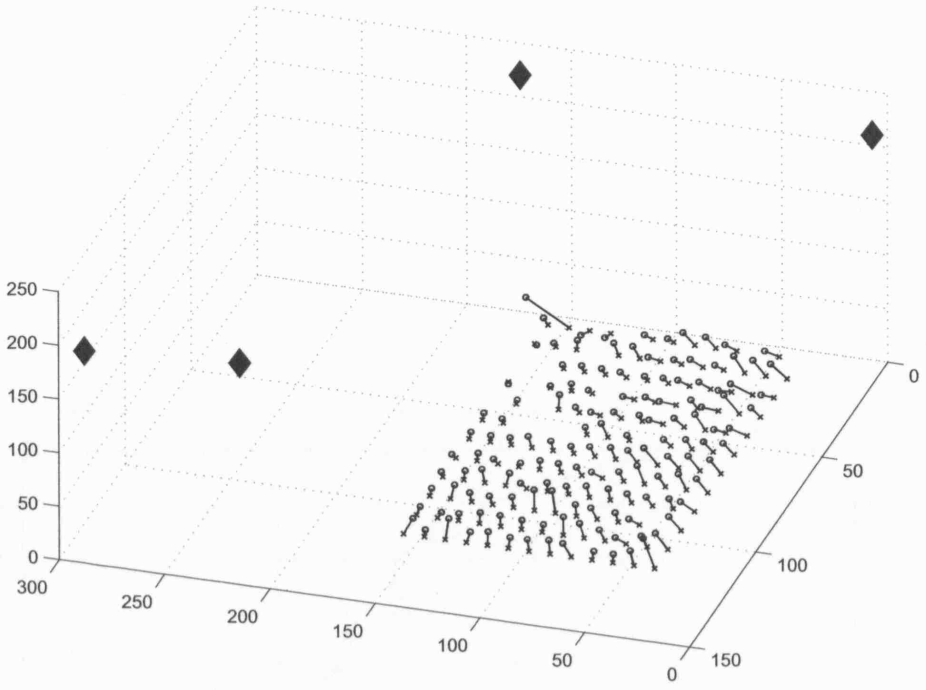
horizontal plane is shown in Figure 5.7.

In an actual implementation the position may be computed on the mobile node itself, by performing the hyperbolic multilateration computations using the sensed beacon signals and a preloaded database of beacon codes and the corresponding (x, y, z) coordinates of the speakers transmitting them. Alternatively, the position information of the speakers may be contained within the beacon signals, although this would require more processing on the part of the mobile device and the use of longer code sequences. Either way, the mobile device's location would not be known to the network, so this may be considered a mode of operation that preserves the privacy of the user. A non-privacy or tracking mode would have the mobile device transmit the recorded beacon signals back to the sensing infrastructure, using a wireless channel such as 802.11 or Bluetooth, for the location computations. This mode sacrifices privacy in favor of hardware simplicity, as it shifts the computational burden from the tracked object to the sensing infrastructure.

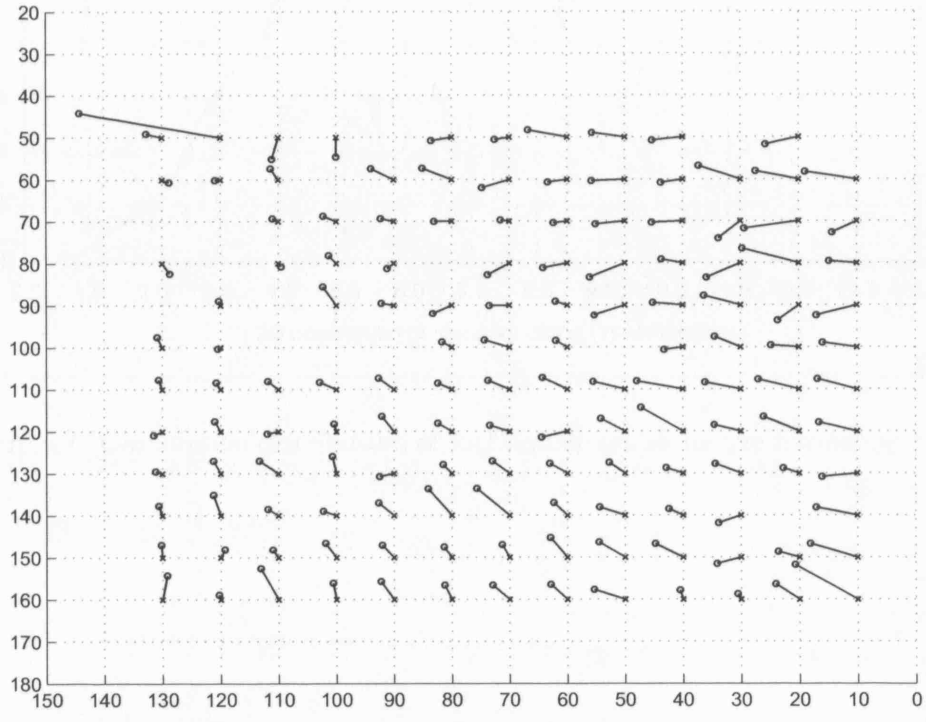
The system described here can be rapidly deployed and used, as no additional hardware construction is needed. The technique does not require any synchronization between the beacons and the located object, and allows very simple commercial devices with little or no computational power, such as an analog wireless microphone, to be tracked. Even an ordinary audio recorder, for example, may continuously record acoustic beacons as it moves within an area, and its traversed path may later be post-processed and reconstructed. However, in an actual working deployment, the use of audio beacons in the audible range might be annoying to users. In such cases, it would be necessary to shift the working frequencies to the ultrasonic range, and although this would require some simple hardware modifications, the basic principles would remain the same.

5.4 Related work

This section provides a brief review of some related work in sensing flow context. However, to put this review in perspective, although three representative models of flow context sensing techniques and their respective implementations were presented, these should be viewed as example components of a larger, integrated conceptual whole. Nonetheless, in the next few sections an attempt is made to compare these individual implementations to existing, related work.



(a)



(b)

Figure 5.6: Positioning results showing actual (x) and computed (o) positions. Axes in centimeters. (a) 3D plot showing beacon locations (◊). (b) Top view.

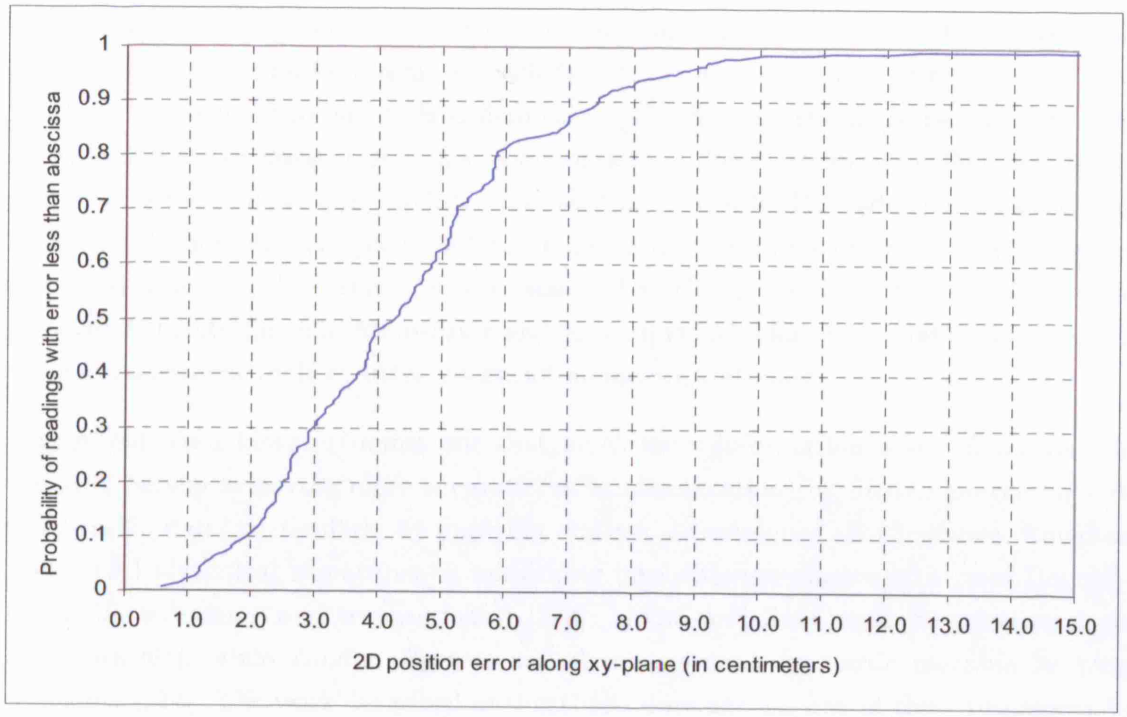


Figure 5.7: Cumulative distribution of location errors along the horizontal plane.

5.4.1 Intrinsic flow context sensing

The concept of "intrinsic flow context sensing" in this thesis may be traced to the work on Internet traffic classification, as the problem of classifying traffic into flows usually involves an examination of the individual PDUs (packets) in the aggregate, the extraction of common features, and their subsequent mapping to sets called flows. Early work by the network measurement community focused on classifying packets into flows for measurement purposes, thus the flow context information of interest was usually limited to the (1) classification parameters, typically based on packet header information, and (2) the metrics, such as flow rates, flow volumes, or packet distributions. Flows were typically classified based on IP protocol, source IP address, source port, destination IP address, and destination port information found in packets [42], or on pairs or lists of netblocks [43] or on other more general parameterized definitions [42]. Sensing traffic characteristics using these parameters, or using clusters and combinations of these parameters called *dimensions* [182] was the focus of metering architectures such as NeTraMet [48] (which in turn used the IETF's RTFM architecture [49]), and still remains the main task of present-day metering systems such as NetFlow from Cisco Systems. The EU-IST MOME Project (Cluster of European Projects aimed at Monitoring and Measurement) maintains a database of similar traffic measurement tools at http://www.ist-mome.org/database/.

More recent work however points out that much more information about flows may be inferred (thereby achieving more accurate traffic classification) by further processing collected traffic data, particularly by applying statistical techniques. For instance, Roughan et al. used statistical signatures to map flows into different classes of service (interactive, bulk, streaming and transactional) [213], Moore and Zuev used Bayesian analysis techniques [64], while Zander, Nguyen and Armitage used stochastic machine learning techniques [214]. The work described in this thesis does not use any of these techniques in processing per-flow traffic data, although the modular nature of `FlowSensor` and the multidimensional sensing philosophy advocated in this thesis suggest that these be investigated in the future for potential integration into `FlowSensor`.

Another trend in recent work is to go beyond the information in the headers and look at the flow's content. This includes work that employs signature scanning on flows to detect a particular application, such as peer-to-peer traffic [52, 53], or to discriminate within a specific application class, such as in spam email filtering [54]. There are also applications that examine flows using signatures, but focus on a particular purpose, such as network intrusion detection [55, 56], or network protection through traffic filtering and shaping [51]. The work by Moore and Papagiannaki on *content-based classification* dealt with the issue of classification itself, rather than on any particular application of it [215]. None

of these existing works however attempt to leverage the work done in other application communities in defining signatures. In contrast, the reuse of existing signatures from different application communities is advocated by the work in this thesis, and is reflected in `FlowSensor`'s design.

Other related work on signatures focus on their automatic *generation* from suspicious traffic [183, 216, 217, 218]. While this was not examined in this thesis, it does suggest the possibility of using ontology-based reasoning (to be discussed in Section 6.3.2) for automatic signature generation in future work.

While the focused and detailed work outlined in the literature have contributed numerous useful techniques for intrinsic context sensing, the work in this thesis is different in the sense that it tries to look at a "bigger picture" that attempts to integrate all of these different techniques and different types of context information in a "cross-layer, cross entity" way. On the other hand, the work in this thesis shares a common philosophy with the approach adopted by Karagiannis, Papagiannaki and Faloutsos, who adopted a "multilevel approach to looking at flows" [184] by correlating flow content (inferred from signatures) to the inferred social and functional activities of the end-hosts with which they are associated. However, Karagiannis et al. relied totally on information observed from the flow itself and did not use sensors that can obtain information directly from extrinsic entities.

In fact, there do not seem to be other existing *flow context* sensing infrastructures designed to sense a wide variety of contextual information as part of an integrated whole. While the existing work in sensor fusion (e.g. [219, 220]) and in context-aware applications [11, 10, 12, 13] certainly made use of multiple types of sensors to obtain context information, the intended consumer of the sensed context information is different in this thesis. The approach taken here is quite forward-looking, as it anticipates a future where different types of sensors embedded in the landscape can be put to effective use, not only by end-applications, but also by context-aware networks.

5.4.2 Location sensing

For location sensing, Girod and Estrin used acoustic spread-spectrum techniques in a ranging system that used frequencies in the audible range [210]. Although their system only produced range (distance) information rather than location, it may be extended through multilateration in order to estimate location. This has in fact served as one of the technical bases for the acoustic location sensing scheme presented here. In addition, their philosophy of using common off-the-shelf hardware inspired the effort in this thesis to design a sensing

system that only uses off-the-shelf devices.

Hazas and Ward presented a privacy-oriented location system based on ultrasonic spread-spectrum and pseudoranging techniques [221]. Although the physical sensing base is similar to the work described in this thesis, their use of pseudoranging has some implications in the overall design of the location system, particularly in the need for synchronization within the system. A pseudoranging system typically requires beacons to be tightly synchronized with each other, and there is likewise some benefit in synchronizing the located object with the beacons as well, as this minimizes the magnitude of the clock bias that needs to be estimated. In a hyperbolic multilateration system similar to the work described here, tight synchronization is required only among pairs of beacons to provide accurate TDOAs, and to a lesser extent, across different pairs. While synchronization between the beacons and the located object may minimize the number of acoustic data samples that need to be processed, the algorithm itself does not require it.

In some cases existing network devices may be used to sense location using data from their associated communication channels. For example, RADAR uses the received signal strength (RSS) data of wireless access points (APs) in order to estimate the location of a wireless mobile device [205]. Cellocate uses the difference in arrival times of a mobile phone's signal at nearby cell sites in order to estimate the phone's location [208]. These systems however are only able to offer *coarse-grained* localization, that is, they are not able to resolve locations to a fine degree of precision [205, 208]. Thus, unlike the system presented in this thesis, it would be difficult to use such coarse-grained systems to sense and resolve locations within small spaces such as rooms.

Other examples of systems used for sensing location in context-aware systems include the Global Positioning System (GPS) [222], the Active Bat Location System [206] and Cricket [207]. Various other location context sensing-systems are surveyed in [200].

5.5 Chapter summary

A basic functionality needed in a context-aware network is a mechanism by which context information, such as flow context, may be obtained. This chapter presented the design and implementation of three different sensors for flow context: a sensor for intrinsic flow context called `FlowSensor`, a sensor for node and device characteristics (a form of extrinsic flow context), and a sensor for node location (another form of extrinsic flow context). The design of each of these sensors contributes functionality and sensing techniques that may be useful in future context-aware networks. In addition, the design of each one provides

some unique insights in terms of approaches and philosophies that may be adopted in the implementation and deployment of other sensors in the future. These approaches and solutions include multi-dimensional sensing, resource reuse, and rapid deployment, which are explained further in the next few paragraphs.

A *multi-dimensional approach* to sensing context, taking into account the multi-faceted nature of flow context, is advocated in this thesis. This was demonstrated in the design of `FlowSensor`, a modular application that combines some of the basic functions found in conventional flow metering architectures, as well as a signature-scanning functionality often found only in specialized applications such as intrusion detection systems and spam email filters. In addition, a design approach that intends to reuse the significant body of tested signatures available within different communities was adopted. `FlowSensor`'s modularity and potential for reuse in other applications was demonstrated by quickly assembling a flow context visualization tool, which may be used in network management.

The location sensor presented in this chapter offers a novel approach to sensing location, using acoustic spread-spectrum techniques. To implement this sensor, only off-the-shelf components were used. In terms of the "bigger picture," this reflects what would be a good approach to building new sensing functionality: try to *reuse available resources*, possibly in a creative way. This philosophy has been inspired by similar approaches to building location systems in the literature, such as in RADAR [205] and Cellocate [208].

In some cases however, the *flexible and rapid deployment* of sensors may be a major issue, especially with heterogeneous and dynamically-evolving networks. In this regard the device sensor described in this thesis contributes a potential solution for rapid deployment, using the active networks paradigm. The application of this deployment technique to the other sensor designs presented in this chapter is planned for future work.

Chapter 6

Aggregating and Disseminating Flow Context

The preceding chapter tackled the issue of sensing flow context within the network and in other relevant entities. As in the previous one, this chapter attempts to answer some questions, but instead of one, it presents three: If sensors can provide flow context, how can they be found? How can the information be disseminated to the entities that need it? What kind of processing has to be done on the information so that it is useful to these entities? These are the issues of locating, disseminating and processing flow context information, first introduced in Section 3.4.2, and discussed in the rest of this chapter.

It is sometimes difficult to cleanly separate the sensing, dissemination, processing and aggregating functionalities from one another, as one stage is often intimately related to or dependent on the next. For example, a sensor that provides per-flow bandwidth consumption information would actually need to sense information on a per-packet level (such as what is done in `FlowSensor` in Section 5.1) and process this sensed information over a certain period of time to provide both instantaneous and average bandwidth figures. Similarly, the dissemination model may also be tied to the aggregation model: an architecture that has a centralized context processing or aggregation function would logically have to disseminate the aggregated, "higher-level" context from that centralized aggregator, leading to a centralized dissemination model, at least for the post-processed context. Nonetheless, in the following sections an attempt is made to discuss each aspect separately. When applicable, frameworks for classifying the different approaches, which can be used to analyze existing approaches in the literature, are provided. More importantly however,

these frameworks may also be used to describe some of the approaches developed and evaluated in this thesis for the problems of location, dissemination and aggregation. These are presented in detail in the next few sections.

6.1 Discovery and location

Before flow context can be disseminated, processed or used, sometimes it has to be *located* first. The issue of knowing where to obtain flow context can equated to the issue of locating the entities that provide context information, such as the sensors themselves, or other entities that process it into a useful format, or entities whose role is primarily to disseminate and make the information available. This is the issue of discovery and location.

If the provisioning of flow context information within a network are considered as a *service*, then one may evaluate and use some of the existing *service discovery protocols* such as those described or supported by the IETF Service Location Protocol (SLP) [223], Jini [224], Universal Plug and Play (UPnP) [225], Universal Description Discovery and Integration (UDDI) [226], the Salutation Protocol[1] [227], and a host of other frameworks, for this purpose. However, the objective in this section is not to evaluate these – the reader may refer instead to [228, 229, 230, 231, 232] for some comprehensive reviews – but rather, to focus on and evaluate a specific approach to locating context sensors that had been suggested in [109, 233].

6.1.1 A distributed approach to locating context sources

Some of the existing service discovery protocols, such as Jini and UDDI, rely on centralized directories or registries to store service information [229, 232]. On one hand this approach offers a relatively simple solution that may work for small-scale implementations. However, as the number of clients increases, such architectures, if they do not resort to strategies such as replication, may start to become bottlenecks and single points of failure [234]. For large-scale, heterogeneous, mobile and highly dynamic networks, it is difficult rely on the presence of fixed, centralized structures for such service directories or registries. On the other hand, a scalable, distributed and self-organizing mechanism would be preferable.

In the Ambient Networks (AN) project (cf. Section 2.3.2), the need for a mechanism with scalable and distributed properties for the purpose of locating context sources was

[1]The Salutation Consortium, which was in charge of developing this protocol, was dissolved on June 30, 2005.

likewise identified, and a solution based on the use of distributed hash tables (DHTs) was suggested [109, 233] but not thoroughly investigated. The next few sections describe a way to implement this mechanism, not only as it applies to ambient networks, but as it may apply to other context-aware networks as well. The design of a simulator developed to evaluate this approach, as well as some experimental results are discussed.

The Context Coordinator (ConCoord) in ContextWare

In ANs, the infrastructure that provides for context management, mediating between information sources such as context sensors and context clients is called ContextWare [109, 31]. The two main functional entities within ContextWare are called the Context Management (CM) Functional Entity, which performs internal management of context information, as well as the associations between context sources and sinks; and the Context Coordination (ConCoord) Functional Entity, which deals with indexing, registering, authorizing and resolving context object identifiers into locators.

The ConCoord supports a number of operational *primitives* to realize its basic functions. A `REGISTER` primitive allows a context sensor to register context object identifiers, called Uniform Context Identifiers or UCIs [233]. A `RESOLVE` on the other hand requests a list of sensor addresses corresponding to a UCI, which are returned to the entity issuing the request through a `RESOLVEACK`. A context consumer then obtains information directly and synchronously[2] from the sensor using the latter's resolved address through a `GET`.

A `SUBSCRIBE` refers to a request for the asynchronous delivery of a certain piece of context information (data or an event) in the future, when it becomes available. `NOTIFY` refers to the actual asynchronous delivery of information, particularly about an event.

Locating context sources through DHTs

A distributed hash table (DHT) is a large hash table cooperatively maintained by multiple nodes in a decentralized and autonomous fashion [235]. Each node manages one or more partitions of the hash table's key space K called *zones* and maintains direct connections with other nodes called *neighbors*. An example of a DHT protocol is the Content-Addressable Networks (CAN) design, which uses a d-dimensional Cartesian coordinate system on a d-torus to map out the key space K for all $<key, value>$ pairs to nodes in the overlay [236]. In CAN, a new node joins the DHT overlay by contacting a nearby

[2]The terms "synchronous" and "asynchronous" are defined in Section 6.2.1 on page 134.

node already in the overlay, and obtains a chunk of the key space by randomly selecting an existing node and requesting half of that node's zone. The new node also builds up its list of neighbors, which are nodes that have zones that are logically adjacent to its zone. When the new node and its neighbors have updated their respective neighbor lists, routing within the DHT overlay now includes the new node. Routing typically uses a greedy algorithm on a hop-by-hop basis. Further details of the operation and maintenance of a CAN may be found in [236].

Preliminary evaluation

The functionality of a context sensor registry (e.g. ConCoord) may be realized by implementing a basic CAN-based DHT and mapping the defined ConCoord primitives `REGISTER` and `RESOLVE` to the counterpart operational primitives `get` and `put` within the DHT.[3] A `REGISTER` may be implemented as a `put(contextID,srcAddr)` in the DHT, while a `RESOLVE` may be implemented as a `get(contextID)`.

With this mapping, the performance of the ConCoord primitives `REGISTER` and `RESOLVE` would ultimately depend on the performance of the `get` and `put` primitives in the underlying DHT. Both the `get` and `put` operations in the DHT require the requesting node to contact a known node in the overlay and have the overlay internally route the requested operation to the node that owns the zone to which the hashed key of the $<key, value>$ pair maps. This is done by message passing on a hop-by-hop basis within the DHT, using its own routing mechanism.

Figure 6.1 illustrates the steps needed to `REGISTER` a context source (e.g. a sensor) located at address `srcAddr`, providing context information with unique identifier `contextID`. The source first contacts a node in the DHT and issues a `put`, supplying its own address and the identifier of the context information it can provide. The DHT node then applies a hash function to `contextID`, the result of which maps to a unique node within the DHT. It then forms a message addressed to that DHT node containing the information to be stored (the `srcAddr` of the context source). The message is then routed on a hop-by-hop basis within the DHT until it reaches the target destination node, which then stores `srcAddr`.

To find context sources that can provide information of type `contextID`, a prospective client needs to `RESOLVE` their addresses via the DHT. The source issues a `get` to a nearby DHT node, supplying `contextID` as parameter. The DHT node applies the hash function to `contextID`, yielding the identifier of the DHT node that stores the information. The

[3] In this section ConCoord primitives use uppercase letters (e.g. `RESOLVE`), while primitives in the underlying DHT use lowercase letters (e.g. `get`)

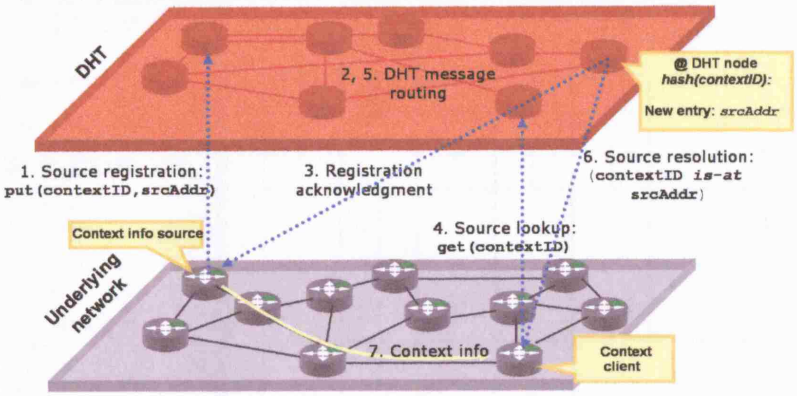


Figure 6.1: Storing context source locations in a DHT

`get` message is then routed to that node, which ultimately retrieves the stored `srcAddr`(es) and sends it to the client that originally made the request.

Based on the foregoing, the performance of the context primitives `REGISTER` and `RESOLVE` would very much depend on the performance of the `get` and `put` primitives in the underlying DHT. One major factor that impacts the performance of the latter two DHT primitives is the number of overlay hops a message would have to traverse from the first DHT node contacted to the target DHT node storing the data. The more hops a message traverses, (1) the greater the latency between request and reply, and (2) the greater the bandwidth it consumes.

The path length in a DHT is therefore a metric worth investigating. In CAN the path length depends on two variables (among others): the total number of DHT nodes, and the number of routing neighbors per node [236]. With an increase in the total number of nodes, there should be an increase in both the maximum and average path length between any two nodes. On the other hand, to increase the number of routing neighbors per node, the number of dimensions d used in constructing CAN's d-torus should be increased as neighbor relations are built on the basis of zone adjacency along a dimension: in other words, more dimensions means more chances of gaining neighbors.

CAN simulator

To validate these claims experimentally with a reasonably large number of nodes, and to facilitate the collection of statistics, software simulator for a DHT that uses the CAN protocol was developed.

Figure 6.2: Screen capture showing `DHTBrowser`'s user interface

`DHTSim` is written for Java 1.5 and serves as the main class for the simulator. It operates in two modes: an interactive mode and a batch mode. Interactive mode, shown in Figure 6.2, provides a graphical view of the CAN DHT and allows the user to interactively add or delete nodes, view per-node zone and neighbor information, and trace message routes within the DHT. Interactive mode can only be invoked when the CAN is 2-dimensional, and is intended more as a tool to visualize the topology and check the correctness of the underlying processes. Interactive mode functionality is provided by the `DHTBrowser` class.

`DHTSim`'s batch mode bypasses the creation of the interactive GUI and automatically builds a DHT based on user-specified parameters such as the number of bits per dimension, an initial number of dimensions, and an initial number of nodes. The software then computes the pairwise routing lengths between all node combinations, logs the output statistics, increments the parameters, and repeats the process.

Figure 6.3 illustrates the functional blocks of the DHT simulator. The `DHTBrowser` interactive mode graphical user interface draws the topology and zone distribution of the 2-d CAN by accessing an internal node table. Creation and deletion of new nodes are also

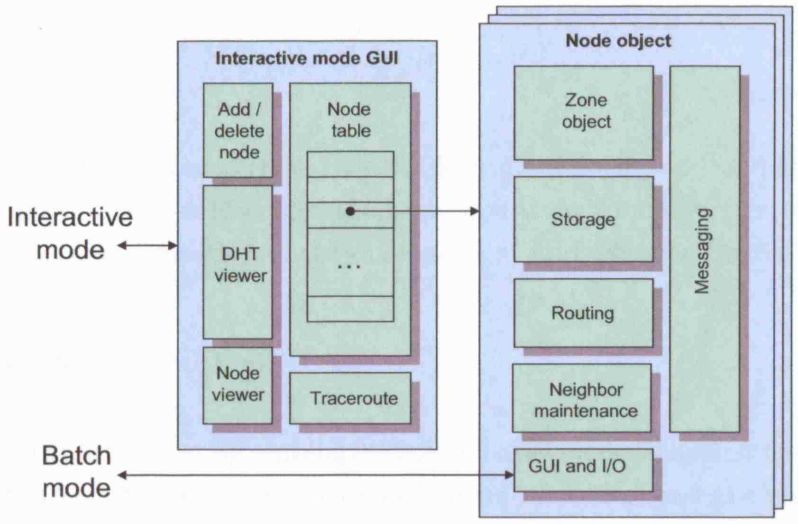


Figure 6.3: DHTSim's functional components

done through this node table. When a user requests to display the properties of a specific node, DHTBrowser calls the method within the Node object itself. Each Node object also has internal methods for storage, routing, neighbor maintenance and inter-node messaging, and maintains a set of Zone objects. Zone objects in turn have their own internal methods to maintain their own boundaries and the ability to split their respective zones in order to allocate portions to newly-created nodes. Zones are internally represented through an ordered pair of boundaries $<leftBoundary, rightBoundary>$ along each coordinate axis. The "left-ness" and "right-ness" of these boundaries are significant as spaces may wrap around the maximum numerical values of the axes (recall that the CAN keyspace K is a d-torus).

In addition to zone maintenance and zone-splitting methods, Zone objects also contain methods to determine adjacency with and distance to other zones. The adjacency method is a service used by nodes to determine if another node is to be included in or excluded from its list of neighbors. The distance computation service allows a node to determine, for routing purposes, which of its neighbors has the least distance (and would thus be the next message hop) to a target zone. The current implementation has a number of "plug-in" methods for computing distances between zones, each employing a different heuristic. These include the various norms of the Minkowski distance of order p (p-norm distance) [237], given by

$$S_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \tag{6.1}$$

such as the Manhattan distance ($p = 1$), Euclidean distance ($p = 2$) and the Chebyshev distance ($p = \infty$). These different methods were written to support future studies on routing performance as a function of the different next-hop selection heuristics.

Simulation results and outlook

The simulation was conducted by running `DHTSim` in batch mode. The first iteration started by creating a CAN with two dimensions and four initial nodes, and the number of hops between all pairs of nodes were calculated. The next iteration incremented the number of nodes, created a new CAN, and performed the same calculations. When a CAN size of 128 nodes was reached, the number of dimensions was increased, and process repeated, starting with four initial nodes, and progressively incrementing the number of nodes until a maximum of 128 nodes was reached. An upper limit of six dimensions was used in the simulation.

The simulation results indicate that a CAN DHT would perform reasonably well as a distributed repository of context source locations for a moderately-sized network of up to 128 nodes. Figure 6.4 shows both the average and worst-case latency of the `get` operation in terms of the number of application-level hops (i.e., routing path length) within the DHT overlay. At the maximum simulated size of 128 nodes, the routing path length for a 4-dimensional CAN is 2.88 hops on average and 6 hops maximum. As with the basic CAN design, the routing path length is influenced by the number of dimensions d used in setting up the key space, since for a given number of nodes n in an overlay, a key space set up with more dimensions would result in a larger number of neighbors per node than a space having fewer dimensions. A larger number of neighbors per node would result in a shorter path length on average, but would require each node to maintain more state. In the results shown, there seems to be a significant improvement both in the average and worst-case `RESOLVE` latency with an increase in the number of dimensions from $d = 2$ to $d = 4$. On the other hand, no significant improvement was observed when the number of dimensions was increased from $d = 4$ to $d = 5$ for networks up to this maximum size. Both of these are consistent with the average path length growth $O(dn^{1/d})$ predicted in [236].

This initial experimental attempt to evaluate the use of a CAN DHT in context dissemination validates the basic CAN performance claims in [236], and demonstrates the feasibility

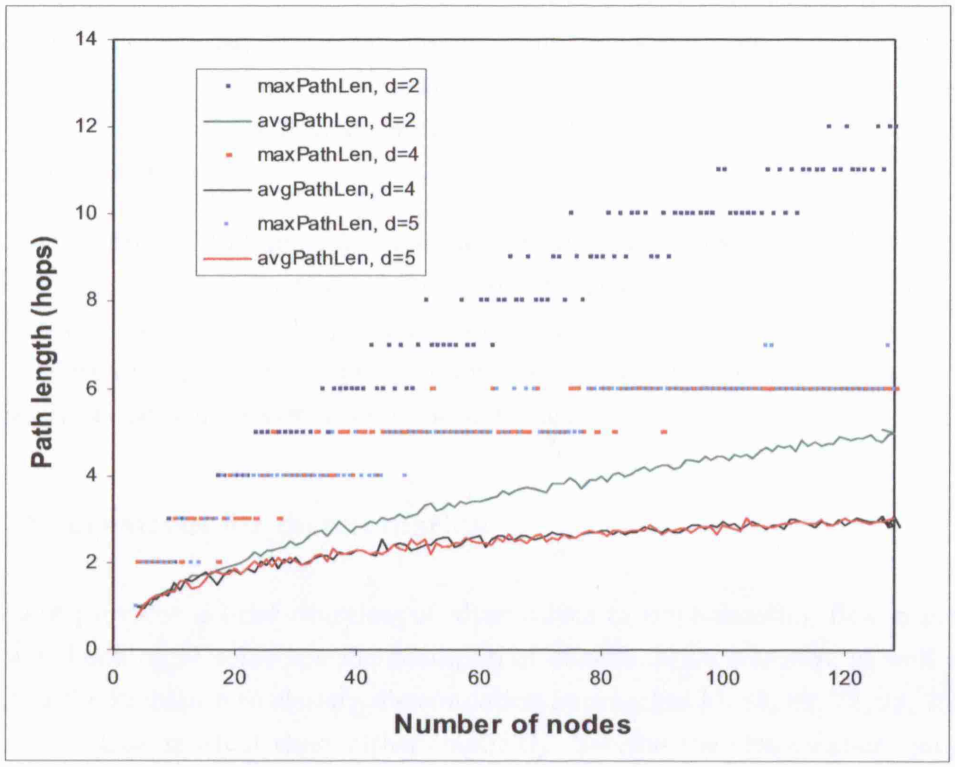


Figure 6.4: Effect of CAN DHT overlay size and dimensions on path length. The color-coded dots in the upper regions of the plot indicate the maximum path lengths obtained for a CAN DHT with d dimensions and a number of nodes n indicated by the x-axis. The continuous graphs indicate the average path lengths obtained for the same set of parameters d and n.

of using CAN-based DHTs as a distributed and scalable mechanism to store and resolve context sensor locations. Further enhancements to the work described here are outlined in Section 8.3.2.

6.2 Context dissemination

Once context sources have been discovered and located, or if their locations are known *a priori*, the next step in the life cycle consists of disseminating context information. Note that the dissemination process may encompass both raw and processed (aggregated) flow context, so in some cases it may be intimately tied to both the sensor deployment and aggregation schemes used.

This section focuses on the dissemination mechanism, and discusses a mechanism tailored particularly for the dissemination of flow context: *path-coupled dissemination*. It also revisits the work on DHTs in the previous section, and reuses it for *path-decoupled dissemination*. Before discussing these specific approaches, next section first presents a general framework for classifying dissemination mechanisms.

6.2.1 Alternatives for dissemination

This section provides a brief overview of alternatives in implementing flow context dissemination. Looking at some specific examples of dissemination schemes, as well as some attempts in the literature to classify dissemination approaches [6, 86, 89, 72, 93, 233, 238], it can be seen that most of these either implicitly describe the classification parameters by citing examples, or do not characterize the approaches using a general classification framework.

The following framework for classifying or characterizing context dissemination schemes based on a more generic description of each axis of classification was thus devised. Although it has been made as broad as possible, it is by no means exhaustive. However, it can be used to classify most of the existing dissemination schemes. More importantly, it provides a framework for describing some dissemination schemes presented later in this section.

Axes of classification The various dissemination schemes may be classified according to:

- *Initiating entity* - this axis of classification refers to the entity that initiates the transfer of context information. If the context consumer initiates the transfer, it is called a *pull*, otherwise if the context source initiates the transfer, it is called a *push*. It is also possible for a consumer to initiate a request for information in advance, or *subscribe* to information that a context source may later asynchronously *publish*, or provide via a mechanism called *callback*. The term usually used for a similar mechanism that involves information about the occurrence of events is called *notification*.

- *Level of temporal coupling* - this refers to the time relationship between the context sources and context consumers. If a source makes a request for context information and normally expects a response within a predetermined maximum amount of time, or blocks other transactions until that particular transaction is fulfilled, the sources and consumers are closely coupled in a temporal sense, and the transactions are said to occur in a *synchronous* way. This definition may also apply if transactions occur with reference to some global timing scheme, since although the coupling is now indirect, the temporal relationship still exists.

  Conversely, if context sources provide information without taking into account any timing relationship or dependency with the consumers (or potential consumers), or, if consumers can receive information at any time, the dissemination scheme is called *asynchronous*. Message queuing systems may be considered forms of asynchronous dissemination; storage may be considered a rather extreme form of asynchronous dissemination.

- *Level of spatial coupling* - this refers to whether the context sources and consumers communicate in a *direct* manner, or if there are one or more intermediaries, resulting in *indirect* communication. Some architectures are called *indirection infrastructures* as they are purposely designed to indirectly link communicating parties via *rendezvous-based communication* [239]. Mechanisms that employ shared spaces for communication rather than direct communication, such as *tuple spaces* [238, 240] and *blackboard systems* [241] may also be considered forms of indirect communication, although it should be noted that the latter two examples refer not only to communication schemes but also to specific distributed computing paradigms.

- *Level of control coupling* - if the dissemination scheme predominantly employs message exchanges that involve data, and the entities do not exercise a large degree of control over each other, the entities are said to be weakly coupled from a control standpoint. On the other hand, it may be possible for one entity to be directly invoking methods within another entity as part of the dissemination scheme (e.g. remote

method invocation or RMI), or for some code functionality from the dissemination architecture to be directly embedded into the sources and consumers of information (e.g. Jini). In this case there is a significant degree of control coupling used within the dissemination architecture. Perhaps an intermediate degree of control coupling is represented by *object* sharing and passing, as this involves not only the passing of data but also methods (via embedded code).

- *Service roles of disseminating entities* - if an entity's primary role is to fulfill service requests related to the dissemination mechanism, then it is acting as a *server*. Conversely, if an entity's primary role is to make such service requests and benefit from the fulfillment of those requests, then it is a *client*. A dissemination architecture that predominantly uses server and client entity roles is called a *client-server* architecture.

  On the other hand, if the entities involved in the dissemination mechanism have both client and server functionalities, and are intended to play both roles (more or less) equally and uniformly, then the dissemination scheme employs a *peer-to-peer* [242] mechanism.

- *Level of decentralization* - dissemination schemes may also be classified in terms of the level of centralization or decentralization of functionality, entities or resources. If the architecture relies only on a few (or perhaps one) entity to perform functions or services that are necessary for dissemination, then it is *centralized*; if these functionalities, services, resources or active entities are distributed throughout the network then it employs a *decentralized* approach.

- *Message scope and routing* - if a single transmission from a source can be simultaneously received by a multiple number of context consumers, the dissemination scheme employs *multicast*. A broadcast is a special type of multicast where the message is sent to all nodes. Otherwise, if a source can only transmit to one consumer, it employs *unicast*. In large and highly decentralized dissemination architectures, the routing protocol used to forward messages from one dissemination entity to the next may also be used as a basis for classification.

6.2.2 Path-coupled and path-decoupled dissemination

The classification framework presented in the previous section depended largely on existing dissemination and communication schemes found in context-aware computing, distributed computing and mobile networking. However, due to the novelty of flow context, a new axis is introduced for characterizing the dissemination of flow context: whether a scheme uses a *path-coupled* or a *path-decoupled* approach. The terminology comes from the work

by the IETF Next Steps in Signaling working group [243], and although the equivalent concepts presented in this thesis were developed independently from the NSIS' work[4], the NSIS terminology was adopted in anticipation of its recognition as a standard.

Path-decoupled flow context dissemination is defined as the distribution of flow context independently of the path traversed by the flow. This means that the flow of context information either does not follow the same path as the flow itself, or if it does, the traversal of that same path is unintentional. Conversely, path-coupled flow context dissemination refers to the distribution of flow context information along the flow path. Like path-coupled signaling in NSIS, path-coupled flow context dissemination is inspired by the Resource Reservation Protocol signaling model, where messages request QoS for a data flow from the routers on the path [45, 243]. It likewise aims to signal information about a data flow along its path for more than just QoS purposes, and for use by different entities within the network [243]. However, unlike the work in NSIS, which focuses more on the architectural framework and protocols for the signaling process itself, this work develops and models the *information about the flow* itself that may ultimately be transported using the NSIS signaling framework.

The next few sections further discusses why these dissemination methods are suitable for flow context.

Path-coupled flow context dissemination

Why use path-coupled dissemination? Handling flow context poses some challenges, because flows are not permanent entities or structures within the network. Flows are ephemeral, that is, they may simply appear and disappear within the network. In some cases, the flows may be short-lived, such as in the case of transfers of small amounts of data, or when mobile hosts briefly transit through an access point's coverage region. Thus, like the flows they describe, a specific flow's context would technically exist only while the flow itself exists (although such context may of course be recorded and stored for future use). It would be impossible therefore for a node within the network to pre-subscribe to a *specific* flow's context, if the existence of that flow cannot be known in advance.

It may of course be possible for some network nodes to pre-subscribe to context information describing all or some flows generated by a particular end-host, or flows that traverse a

[4]The author of this thesis published a concept for flow context dissemination similar to path-coupled signaling in a paper for APGAC'05 in May 2005 [74]. RFC 4080, which discusses the NSIS framework, was published in June 2005, although the requirement for path-coupled signaling appeared earlier in April 2004 [244]. The author had been unaware of the NSIS' work until after the APGAC'05 paper.

specific context-sensing middlebox. However, in the case of mobile hosts or ad-hoc wireless networks, this may again pose some complications: since end-hosts or nodes also appear and disappear within the network, it would difficult for another node deep within the network to pre-subscribe to context information generated by a mobile or ad-hoc node that has not yet joined the network. In wired networks, although the location of the nodes within the topology may more or less be fixed, dynamic packet and flow routing may also imply that network nodes cannot predict with certainty if flows generated by or transiting through other nodes will traverse a context-"interested" node. Thus, except in very simple network topologies, network nodes generally would not know in advance where they might obtain context information about flows.

In [74], Ocampo et al. proposed the use of a push-based method called *flow context tagging*. The use of the term in that work is significantly different from the way it is defined in Section 3.3.4 and in the rest of this thesis. In [74] "flow context tagging" referred to a path-coupled, push-based, asynchronous, spatially direct or indirect, multicast mode of flow context dissemination from context sensors to context consumers.[5] In Section 3.3.4 however, and in the rest of this thesis, flow context tagging refers to the act of explicitly encoding the context of a flow and associating it with the flow being described.

With path-coupled dissemination, the sensed context information is injected along with the flow, where it is generally expected, but not necessarily guaranteed (due to possible changes in routing), to traverse the same path as the flow. The context tag is associated with the flow through a fragment of the contextual description that defines the set of PDUs constituting the flow described by the tag, such as through the *filterspec* in RSVP [142], or a subclass of the `Identifier` context information (cf. "Classes and class hierarchies" on page 70), or perhaps through signature specifications (cf. "Signature scanning" on page 100). Path-coupled dissemination provides the following advantages:

1. The push-based approach takes into account the ephemeral and possibly short-lived nature of flows. It does not require context-interested or context-sensitive nodes within the network to know in advance the specific end-host, device or peer node within the network where flow context must be obtained or subscribed.

2. Path coupling offers a focused distribution of flow context: the information accompanies the flow itself along the downstream path, where it is most likely to be used, because that is also where flow impacts the network resources such as link capacity, node processing and computation, and node buffers. Ultimately, the context tags are received at the destination of the flow, where they may also be processed and used.

[5]These terms are defined in Section 6.2.1 on page 134.

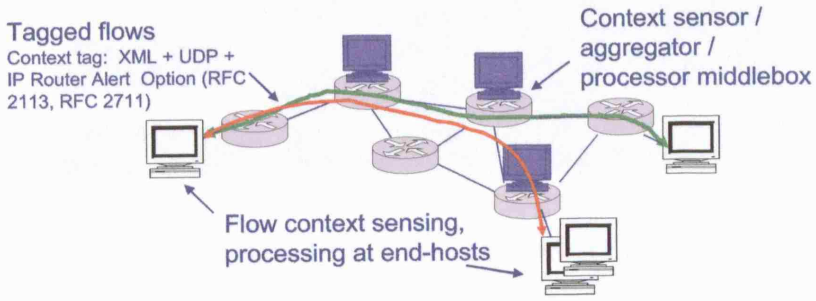


Figure 6.5: Typical deployment of components for flow context tagging

3. The source of the context can inject tags at a rate consistent with the frequency of change in information. In contrast, a client polling information from a context server may not be doing it at an appropriate rate, resulting in *under-sampling* leading to a loss of information.

The use of path-coupled dissemination can be made compatible with persistent context storage, when necessary. If the location of context repositories are known, then end-hosts and context sensors may upload flow context to these repositories. The corresponding context tags could then contain references (pointers) to these context repositories. This provides two useful functions: (1) flow context information may be archived for future use, for whatever purpose it may serve and (2) large chunks of context details may be uploaded to the repository instead of being contained within the tag itself, reducing their size and impact on network traffic.

Mechanics The mechanics of this approach, illustrated in Figures 6.5 and 6.6, are described in this section.

After context information is sensed within an end-host or middlebox, it is passed down to a marshalling and encapsulation function, where it is serialized in XML [100]. At this point the filterspec or the relevant flow identifier or descriptor is also included in the XML-formatted context tag. The vocabulary used in these tags conforms to the terms defined in the ontology in Section 4.3.

The tag is then packaged in a User Datagram Protocol (UDP) transport-layer datagram [245]. The use of UDP allows other context-interested hosts, including the end-destination of the flow, to detect or demultiplex out the context tag packet from the rest of the flow. Further down the processing stack, a tag injection and detection stage encapsulates the UDP datagram in an IP packet whose header contains the IP Router Alert Option as

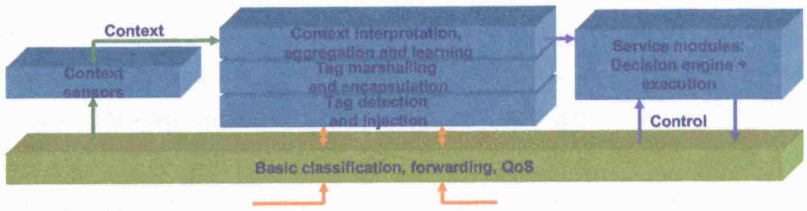


Figure 6.6: Functional components in context tag processing stack

described in RFC 2113 for IPv4 [246], and RFC 2711 for IPv6 [247]. This option has the semantic "routers should examine this packet more closely."

Nodes and devices within the network, particularly routers, detect the context tag by virtue of the Router Alert option in the IP header. Routers that either do not support the option, or do not recognize the context payload simply forward the packet to the next hop. At end-hosts, the context tag may be demultiplexed out of the flow by virtue of the UDP port number. If no equivalent context-processing process exists at the destination host the tag is either silently dropped, or an error message may be returned.

At this point, the node may either re-inject the unmodified tag back into the stream, or process and aggregate the received information (discussed in Section 6.3) before (optionally) re-injecting a new tag. It may also use the context information for flow adaptation, or to trigger or influence the execution of a service, control, or management function. By re-injecting processed context back into the flow, new knowledge may be shared with downstream nodes. Tags like these may also contain information about any adaptation that was performed by upstream nodes on the flow.

A path-decoupled approach using ContextWare

This section provides an example of a path-decoupled approach to flow context dissemination, using components from the ContextWare architecture (cf. Section 2.3.2 on page 30). Section 6.1.1 of this chapter described the role of the ConCoord in locating context sources and evaluated its possible implementation using distributed hash tables (DHTs). In this section the DHT-based ConCoord is revisited, and its functionality is extended for use in path-decoupled dissemination of context events.

As described in Section 6.1.1, ContextWare also has the following defined primitives: `GET`, `SUBSCRIBE` and `NOTIFY`. An entity is needed where subscription requests should be registered; otherwise, all clients that require notification would have no choice but to register subscription requests to *all* potential context sensors that may provide such notifications

– an option which is clearly impractical, and not scalable. To provide notification support in a scalable and distributed manner, the DHT is used as an *indirection infrastructure* for *rendezvous-based communication* [239]. A context client that wishes to `SUBSCRIBE` to an event is mapped to a `put(event,client-address)` in the underlying DHT, where the event descriptor is hashed and mapped to a point in key space K'. For notifications, two implementation options are possible: either (1) the sensor can request a `get(event)`, which returns the addresses of clients subscribed to that event, allowing the sensor to send detailed information about the event to these clients, or (2) a new DHT primitive `send(event,data)` that would cause the DHT itself to send data to the clients subscribed to that event notification could be introduced. Note that `data` may either contain the notifying sensor's address, or detailed event information, or both. Option (2) obviously shifts the actual notification burden from the sensors to the DHT overlay, but in doing so, saves the additional step needed in option (1) for the notifying sensor to retrieve the list of subscribed clients from the DHT. Since the primitives `put(contextID,srcAddr)` and `put(event,client-address)` are used with different semantics, it is proposed that the key spaces K (used for the `REGISTER`, `RESOLVE`, and `GET` primitives) and K' (used for `SUBSCRIBE` and `NOTIFY`) be maintained as separate and distinct spaces to avoid collisions. This can be accomplished by considering them to be separate *realities* in CAN terminology [236].

6.3 Processing and aggregating flow context

Raw context information obtained by sensors and disseminated through the network may have to be further processed and *aggregated* to transform it into a form useful or relevant to the consumers of the information. Processing and aggregation both refer to the wide variety of functions summarized in Section 3.4.1, including data formatting, range checking and data validation, algorithmic computation and normalization, data fusion, augmentation, or the interpretation or abstraction of low-level data into higher-level information. Various techniques may be applied in order to achieve these functions, such as statistical methods, time-domain analysis, neural networks and rule-based logic [11], or the use of artificial intelligence and machine learning techniques [78, 79].

The process of aggregation may transform context information into a form more useful or appropriate to a consumer. Considering that the usefulness, desirability or relevance of context information might actually be modified by changing its accuracy, precision, timeliness, scope and other measures of its quality, then it can be argued that context aggregation may also have an impact on the QoC or quality of context (discussed in

Section 3.3.3). In addition, context aggregation may help minimize the network overhead during context distribution, and help avoid information *overload* at the consumer. If several pieces of context may be aggregated into a more compact and summarized (or abstracted) form that is useful or acceptable to a consumer, then this would mean that the original multiple pieces of context would not have to be transmitted and processed at the other end.

If aggregation is performed without any concurrent processing or computation, this could be considered simple *storage*. However, even the storage of context information may require the data to be formatted or processed prior to or during storage, or during retrieval. For example, flow records might be compressed to minimize memory requirements in flow context sensors, or use formats that minimize their bandwidth overhead during transmission. *Caching* might be considered a special type of storage, where an item is temporarily stored to exploit temporal and spatial locality properties. For example, a piece of flow context information, plus perhaps some semantically related ones, might be cached at a node topologically close to a context consumer, because there may be a high probability that it would be accessed again soon (temporal locality), or that the consumer might also need the semantically-related ones.[6] For example, if a flow context consumer requests information on the current utilization of a network interface, it would be reasonable to hypothesize[7] that it would also need related information such as the type of that interface and the maximum bandwidth it can support.

This section focuses on the high-level *semantic* processing and aggregation of flow context, rather than either the relatively low-level functions of data formatting and validation, or on storage. The different aggregation modes based on their scope are first defined. The novel use of logical inference and reasoning in the semantic processing and aggregation of flow context, including the use of rules and queries, is then explored, and its computational performance is examined.

6.3.1 Aggregation modes

This thesis defines three main modes of flow context aggregation: temporal aggregation, cross-flow aggregation, and cross-type aggregation.

- *Temporal aggregation* involves the observation of a single flow (at any flow abstraction

[6] Although this is quite different from the classical notion of spatial locality in traditional memory caching systems, one could argue that the spatial locality is in the form of some notion of semantic distance such as similarity, logical subsumption, or relatedness.

[7] This has not been investigated in this thesis and might be an interesting topic for future work.

level) or macroflow over time to derive relevant context information. For example, the collection of per-packet statistics to generate information such as flow bandwidth (both instantaneous and average), packet loss, jitter, and related statistical information would require the aggregation of flow-related context over time. Storage and provisioning of historical data would also require temporal aggregation.

- *Cross-flow aggregation* involves the observation of multiple flows or macroflows to form new flow context information. For example, the computation of the total bandwidth consumed by all flows on a link to infer the level of availability or congestion of that link might be considered a form of cross-flow aggregation. Another example might be the examination of the common properties of multiple flows to form new logical macroflows, such as when multiple flows to a single end-host are inferred to be part of a bundle of client-server flows, with the common end-host acting as a server.

- *Cross-type aggregation* involves the observation and processing of multiple, different types of context information to produce new information. A concept very much related to this type of aggregation is *sensor fusion*, which pertains to the combination of data from multiple information sources (sensors) [219, 220]. Other than the distinction that the former concept refers primarily to the combination of multiple data *types* which may or may not come from multiple sources, the two may be considered equivalent enough such that their principles, processes and techniques may held in common.

These aggregation modes are not mutually exclusive, as one may find instances where they are used in combination. It is possible for instance to have temporal aggregation of a single type of context information across multiple flows, from multiple sensors – a simple example of this might be a sensor that collects per-flow bandwidth information, over time, from multiple upstream sources. Similarly, it may be possible (or sometimes even necessary) to aggregate and process different types of context across multiple flows to infer new information. For example, if an active flow $flow_a$ is known to belong to $user_x$, and $flow_b$, containing streaming video, is terminating at the same host where $flow_a$ originates, then it might be reasonable to believe (correctly or otherwise, as it may turn out) that one of the current activities of $user_x$ is `WatchingVideo`.

In order to support inferences of the latter type, it is useful to have an explicit semantic model of the concepts and relationships that define flows and their context, or an ontology, as previously discussed in Chapter 4. In addition, it would also be useful to be able to process not only these abstract concepts and relationships, but assertions or descriptions about specific *individuals* based on the vocabulary defined in the ontology. The ability to

derive inferences from ontologies and assertions about individuals is the role of a *reasoner*, which is discussed in the next section.

6.3.2 Reasoner-based context processing and aggregation

An *automated reasoner* in description logics (DL) is a software program that provides a number of reasoning services based on logic inference. These services include checks for satisfiability, subsumption, equivalence and disjointness for concepts in a TBox[8] through the application of logical inference [248]. Satisfiability may be qualitatively described by the question: "Is the set of objects described by a concept empty?" while subsumption may be described by "Is there a subset relationship between the set of objects described by two concepts?" [249]. Equivalence pertains to whether two concepts describe the same set of objects, while disjointness means that two concepts do not describe any objects in common. A TBox may also be checked for *coherence* or *consistency* by checking if it has any unsatisfiable concepts [176]. In addition, a TBox may be *classified*, that is, the entire subsumption hierarchy or *taxonomy* of concept names may be computed by a reasoner.

A reasoner may also provide inference services for an ABox.[9] The following inference services may be provided: consistency, instance checking, instance retrieval, realization, and role filler computation, all with respect to a TBox [249]. Qualitatively, checking the consistency of an ABox entails checking if all of the assertions in the ABox are satisfiable with respect to the TBox. Instance checking poses the question, "Is the specified individual an instance of the query concept?" Instance retrieval on the other hand finds all such individuals from the ABox that can be proven to be instances of the the query concept.

In ABox realization, the most specific concept names in the TBox that apply to an instance are computed. Finally, role filler computation refers to the retrieval of individuals that are related to a specified query individual by a certain property or relation.

Flow context assertions

This section reuses the same RacerPro [136] reasoner, used for ontology checking in Section 4.3.2, for reasoner-based flow context aggregation. The initial step involves loading RacerPro with the flow context ontology (and its associated imported ontologies). Context information about flows and other entities are then sent to the reasoner as *assertions*. A

[8] A *TBox* contains the set of axioms that define concepts, properties and their various relationships in a knowledge base or KB.

[9] An *ABox* contains assertions about specific individuals in a knowledge base.

subset of the supported ABox assertions in RacerPro, are used in this thesis, including:

- *Concept assertions*, using the `instance` keyword, which state that the individual is an instance of a specified concept
- *Role assertions*, using the `related` keyword, which state that an individual is related via a specified role[10] ("property") to another specified individual, and
- *Attribute assertions*, using the `constrained` keyword, which state that an individual is related via a specified role to an object in the concrete domain [250].

For example, the following assertion, which follows the general format[11] of `FlowSensor`'s output (cf. Section 5.1)

```
(instance |f6-fe92| |L3_IPv4_OneWay_UDP_SimpleFlow|)
(related |f6-fe92| |10.0.1.65| |hasL3_IPv4SourceAddress|)
(related |f6-fe92| |10.0.1.36| |hasL3_IPv4DestinationAddress|)
(related |f6-fe92| |53| |hasL4_UDP_SourcePort|)
(related |f6-fe92| |32776| |hasL4_UDP_DestinationPort|)
```

states that the individual `f6-fe92` is an instance of `L3_IPv4_OneWay_UDP_SimpleFlow`, which in the flow context ontology represents the class of layer-3 unidirectional flows of IPv4 packets with UDP payloads, originating from a single source and terminating at a single destination. Such flows are completely described by the tuple ⟨*source IP address, source UDP port, destination IP address, destination UDP port*⟩. These values are likewise specified in the given example through the use of the `related` keyword. A similar assertion for another flow instance is shown in Figure 6.7, as viewed through the reasoner's graphical user interface.

Rules and queries

Once context information has been fed into the reasoner, inferences can be drawn on the individuals (flows and other entities) modeled in the ABox. As previously mentioned, the inference services typically offered for ABox individuals include consistency checking, instance checking, instance retrieval, realization, and role filler computation, all with respect

[10]The term "role" is a RacerPro-specific term synonymous with property or relation.

[11]The unique flow IDs used by `FlowSensor` are actually 128 bits long. In addition, the RacerPorter graphical interface allows names to be displayed in abbreviated form, that is, for the full namespace URI to be omitted. Both abbreviated forms will be used in subsequent examples for ease of presentation

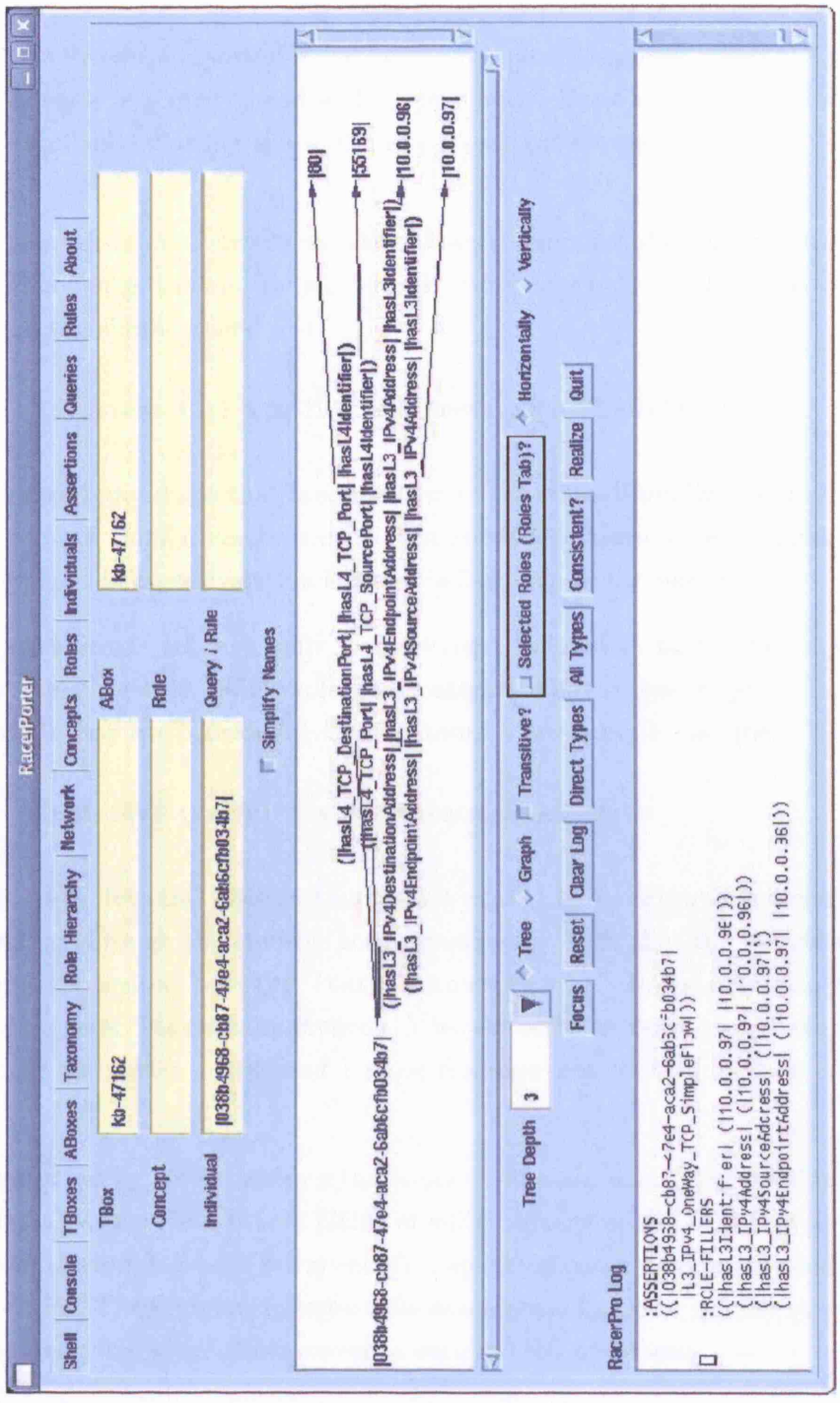


Figure 6.7: A flow context assertion in the RacerPro reasoner

to the flow context ontology. Two facilities provided in the RacerPro reasoner to access these services: queries and rules.

Queries in RacerPro are supported by an expressive query language called nRQL [136, 251]. Each query consists of a query head and a query body. Basic query expressions are called *query atoms*, and the following types of query atoms are supported:

- *Concept query atoms* retrieve the individuals (members) of a concept. Concept query atoms are *unary*, that is, they reference either one individual or one variable. For example, the simple query

  ```
  (retrieve (?x) (?x |L3_IPv4_OneWay_UDP_SimpleFlow|))
  ```

  retrieves all individuals that are members of the class of unidirectional flows of IPv4 UDP packets from a single source to a single destination (as defined in the flow ontology), and successively binds these individuals to the variable `?x`.

- *Role query atoms* retrieve pairs of individuals related to each other through a certain property or role. Since role query atoms reference two objects (individuals or variables), they are considered *binary* atoms. For example, the query

  ```
  (retrieve (?x ?y) (?x ?y |terminatesAtNode|)
  ```

  would return flow and node instance pairs related by `terminatesAtNode`. Roles may be inverted through the `inv` role term constructor, such that the inverse relationship may be represented by `(inv |terminatesAtNode|)`. Roles may also be negated, such that `(not |terminatesAtNode|)` would refer to flows that have either been explicitly asserted to or *proved* by the reasoner not to terminate at the specified node.

- *Constraint query atoms* address the concrete domain part (i.e., the domain of integers, reals, strings, and others [250]) of a KB. Query atoms of this type are used to retrieve individuals whose fillers satisfy a specified concrete domain predicate, called a *constraint*. These are useful especially when reasoning over quantitative assertions, such as querying which flows exceed a certain QoS parameter.

- `SAME-AS` *query atoms* enforce the binding of a variable to an individual. Query atoms of this type are binary.

- HAS-KNOWN-SUCCESSOR *query atoms* belong to a class called *auxiliary query atoms*. HAS-KNOWN-SUCCESSOR retrieves individuals that have explicitly modeled successors on a certain property or role, without returning the actual successor. If the previous example is rewritten as

```
(retrieve (?x) (?x (has-known-successor |terminatesAtNode|)))
```

  the reasoner will return all flows whose termination nodes have been explicitly asserted in the ABox.

Query atoms can be combined using constructors such as (among others) **and**, **union**, and **neg**. The **neg** constructor has *negation-as-failure (NAF)* semantics, where a statement that cannot be proved true is taken as false. This is different from *classical negation*, where a statement is taken to be false if and only if it can be proved to be false.

As an example, the following complex query (called **SearchForPairs**) formed from the conjunction of multiple query atoms, returns TCP flows that form counterpart pairs, where the destination $\langle address, port \rangle$ of one flow is the source $\langle address, port \rangle$ of the other, and vice-versa (i.e. bidirectional "streams" [43]).

```
(RETRIEVE
    (?X1 ?X2)
    (AND (?X1 |L3_IPv4_OneWay_TCP_SimpleFlow|)
         (?X2 |L3_IPv4_OneWay_TCP_SimpleFlow|)
         (?X1 ?A1 |hasL3_IPv4SourceAddress|)
         (?X2 ?A1 |hasL3_IPv4DestinationAddress|)
         (?X2 ?A2 |hasL3_IPv4SourceAddress|)
         (?X1 ?A2 |hasL3_IPv4DestinationAddress|)
         (?X1 ?P1 |hasL4_TCP_SourcePort|)
         (?X2 ?P1 |hasL4_TCP_DestinationPort|)
         (?X2 ?P2 |hasL4_TCP_SourcePort|)
         (?X1 ?P2 |hasL4_TCP_DestinationPort|)
    )
)
```

Rules A rule may generally be defined as a principle or condition that customarily governs behavior.[12] In the context of the Semantic Web, rules may be defined as axioms that

[12] http://wordnet.princeton.edu/

form an implication between an antecedent (head) and a consequent (body), that is, they form an *if-then* relationship between the antecedent and the consequent. Whenever the conditions specified in the consequent hold, the conditions specified in the consequent must also hold [252].

The inclusion or "layering" of rules on top of ontologies in the Semantic Web has been proposed due to the relative weakness of the OWL language by itself in talking about properties [253]. A often-cited classic example is its inability to represent the "uncle" relationship in terms of the "brother" and "parent" relations in terms of other built-in relations such as subsumption, disjointness, equivalence and others, without resorting to an operation like property composition [253]. On the other hand, it would be easier to express the "uncle" concept in the following form:

```
IF hasParent(x,y) AND hasBrother(y,z) THEN hasUncle(x,z)
```

The current proposal in the W3C for a Semantic Web rule language is SWRL (Semantic Web Rule Language) [252]. SWRL rules are of the form (using an equivalent human-readable syntax)

$$\text{antecedent} \implies \text{consequent}$$

where both antecedent and consequent are conjunctions (ANDed combinations) of atoms, expressed as $a_1 \wedge \cdots \wedge a_n$. Variables are prefixed with a question mark (e.g., ?x). For example, the "uncle rule" would be expressed in SWRL as

$$\mathsf{hasParent}(?x, ?y) \wedge \mathsf{hasBrother}(?y, ?z) \implies \mathsf{hasUncle}(?x, ?z)$$

Atoms are of the form C(x), P(x,y), sameAs(x,y), differentFrom(x,y), or builtIn(r,x,...), where C is an OWL description or data range, P is an OWL property, r is a built-in relation, x and y are either variables, OWL individuals or OWL data values.

Although RacerPro supports SWRL on an experimental basis, the built-in facilities for rules in the query language nRQL are used in this thesis. In nRQL the antecedent is simply an nRQL query body; the consequence[13] consists of a set of ABox assertions. nRQL rules therefore allow the ABox to be augmented based on the results of queries.

[13]This is the term used in RacerPro, rather than "consequent"

As an example, the following rule (called `ConsolidatePairsRule`) also looks for counterpart inverse flows as in the previous query example. However, it allows new information to be added to the ABox by defining the two flows to be symmetrically related by a `hasCounterpartInverseFlow` relation if the antecedent is satisfied:

```
(FIRERULE
    (AND (NEG (?X1 (HAS-KNOWN-SUCCESSOR |hasCounterpartInverseFlow|)))
         (NEG (?X2 (HAS-KNOWN-SUCCESSOR |hasCounterpartInverseFlow|)))
         (?X1 |L3_IPv4_OneWay_TCP_SimpleFlow|)
         (?X2 |L3_IPv4_OneWay_TCP_SimpleFlow|)
         (?X1 ?A1 |hasL3_IPv4SourceAddress|)
         (?X2 ?A1 |hasL3_IPv4DestinationAddress|)
         (?X2 ?A2 |hasL3_IPv4SourceAddress|)
         (?X1 ?A2 |hasL3_IPv4DestinationAddress|)
         (?X1 ?P1 |hasL4_TCP_SourcePort|)
         (?X2 ?P1 |hasL4_TCP_DestinationPort|)
         (?X2 ?P2 |hasL4_TCP_SourcePort|)
         (?X1 ?P2 |hasL4_TCP_DestinationPort|)
    )
    (
         (RELATED ?X1 ?X2 |hasCounterpartInverseFlow|)
    )
)
```

Performance

The previous section illustrated how ontologies and flow context assertions may be processed by a reasoner in order to create new context information. While the previous section demonstrates that it *works*, it is also important to determine if it can used in real-world scenarios on platforms that might have limited amounts of processing power and memory, and which have to deal with a large number of flow instances being detected at a high rate.

This section describes a series of experiments that were aimed at studying the performance of reasoner-based context processing and aggregation. To simulate an operational scenario as realistically as possible, actual *packet header traces* were used and transformed into flow instances. Packet header traces are records of actual network traffic, typically collected from a single link or set of router interfaces, containing *anonymized* header information. The process of anonymization either removes or obfuscates some information inside these headers, while trying to preserve the relationships between them, for privacy

and security purposes [254]. A trace available from the National Laboratory for Applied Network Research (NLANR), specifically trace PSC-1144705892-1.tar.gz, dated 10 April 2006, collected from an OC48c link at the Pittsburgh Supercomputing Center was used in the experiments described here.[14]

In order to decouple the performance of the context reasoning component from that of the flow sensing component, and since the traces do not contain packet payloads anyway, `FlowSensor` (cf. Section 5.1) was not used in these experiments. Instead, a small program using some of `FlowSensor`'s components was assembled and used to transform the trace information into flow instance assertions and basic assertions about their intrinsic context, such as their protocol type, source and destination addresses, and source and destination ports (when applicable). These assertions were stored in a file. A sample assertion (which was also used as an example in the previous section) is shown below.

```
(instance |f6-fe92| |L3_IPv4_OneWay_UDP_SimpleFlow|)
(related |f6-fe92| |10.0.1.65| |hasL3_IPv4SourceAddress|)
(related |f6-fe92| |10.0.1.36| |hasL3_IPv4DestinationAddress|)
(related |f6-fe92| |53| |hasL4_UDP_SourcePort|)
(related |f6-fe92| |32776| |hasL4_UDP_DestinationPort|)
```

A starting point in the file was selected and flow context assertions were fed sequentially into the reasoner. The file reader and the reasoner communicated via UDP, but ran on the same physical machine to avoid network latencies. The rate at which the assertions were fed into the reasoner was dictated by the rate at which the latter processed the assertions, including those already in its ABox, with the reasoner signaling back when it was ready to receive a new assertion. Query and rule evaluation response times were then measured as the number of assertions increased. In all trials, the test platform consisted of an IBM Thinkpad T42p with an Intel Pentium M processor 1.70GHz and 1GB memory, running RacerPro 1.9.0 x86 Linux version, on a Linux 2.6 / Fedora Core 4 operating system.

Query performance was evaluated using the `SearchForPairs` query previously given (cf. page 148). Figure 6.8 plots the response time to this query as a function of the number of flow instance assertions. From the data, it can be seen that the query response time is quite poor for applications that require real-time flow context aggregation. For example, in Figure 6.8a, when the reasoner had around 200 flow instance assertions (with the associated context assertions), the response time was around 2868 milliseconds – almost 3 seconds. With this kind of performance it would be reasonable to suggest that this reasoner would be overwhelmed on a moderately-sized network with 40 active users, each maintaining five

[14] http://pma.nlanr.net/Traces/Traces/daily/20060410/

active flows on average, if new flows are being created at intervals less than three seconds apart.

The response times between the first time the reasoner processes a query and any subsequent repetitions of that query, assuming no changes in the ABox in that interval, are likewise compared. Figure 6.8a shows that the response time of *subsequent* queries is significantly lower, suggesting some computational overhead associated with the initial query processing. This appears to be a platform-specific artifact resulting from the reasoner computing what it calls a *substrate structure* for the ABox when it encounters a query for the first time [255]. Although this cannot be generalized to all reasoners, the finding is nonetheless useful as it suggests that in cases where such initial processing overheads do exist, the average response time would depend on the ratio of initial to subsequent repetitions of the query, assuming no intermediate changes in the ABox.

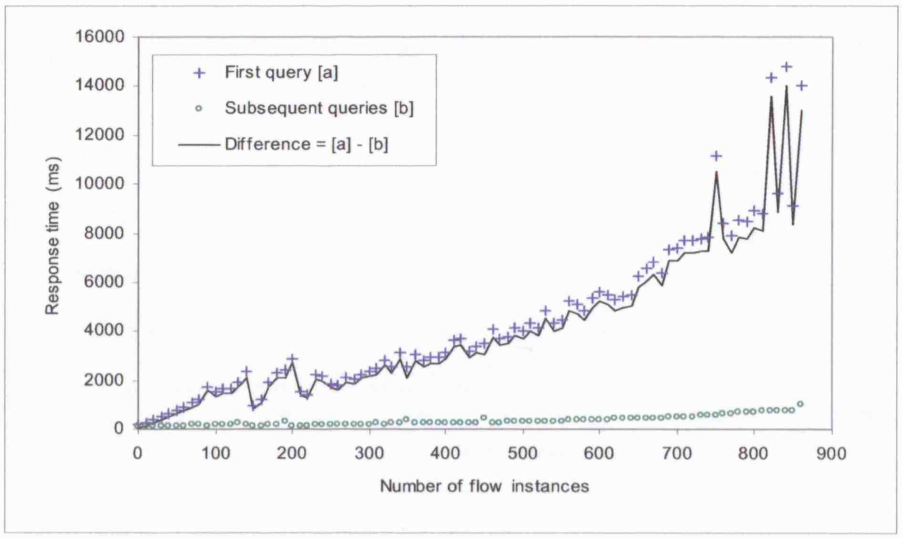
It can also be observed that when the number of flow instances in the ABox starts to exceed 800, oscillations in the response time, followed by a sharp increase at around 860 flows can be seen. This performance degradation was traced to insufficient memory (heap) in the test platform, as shown in the log files.

To verify if this behavior would be found in other queries, `PointToMultipoint`, listed below, was formulated:

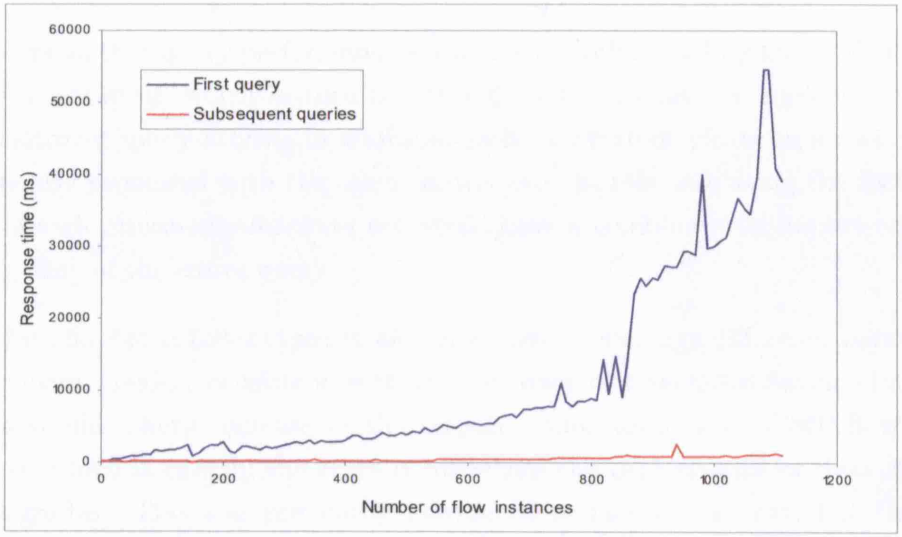
```
(RETRIEVE
    (?X ?A ?P)
    (AND
        (?A (AT-LEAST 2
            (INV |hasL3_IPv4SourceAddress|) |L3_IPv4_OneWay_TCP_SimpleFlow|))
        (?P (AT-LEAST 2
            (INV |hasL4_TCP_SourcePort|) |L3_IPv4_OneWay_TCP_SimpleFlow|))
        (?X ?A |hasL3_IPv4SourceAddress|)
        (?X ?P |hasL4_TCP_SourcePort|)
    )
)
```

This query attempts to find multiple unidirectional TCP flows that originate from the same IP address and TCP port, forming point-to-multipoint flow aggregates, and possibly suggesting server activity on the end-host. A comparison between `PointToMultipoint` and `SearchForPairs` in terms of response times is shown in Figure 6.9.

As can be seen, `PointToMultipoint` performs even worse than `SearchForPairs`. The



(a)



(b)

Figure 6.8: Response time of the `SearchForPairs` query. (a) Up to 860 flows (b) Over the entire range of values, showing the sharp increase at $x = 860$ flows

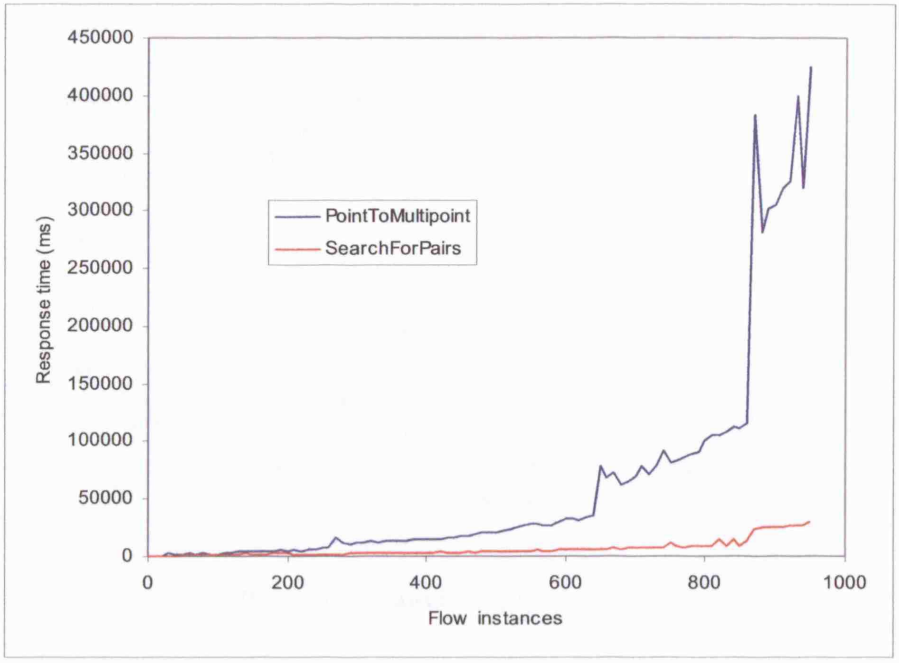


Figure 6.9: Comparative response times of `SearchForPairs` and `PointToMultipoint`

graph illustrates that query performance is (obviously) influenced by the implicit complexity of the query itself, which in turn is dictated by the operations implemented by each of the constituent query atoms. In addition, each query atom yields an answer set which is subsequently processed with the other answer sets, in this case using the `AND` operator. The size of each intermediate answer set would have a combinatorial impact on the total processing time of the entire query.

The raw data for `PointToMultipoint` also indicates a very large difference between initial and subsequent queries, consistent with the behavior of `SearchForPairs`. Interestingly enough, a similar sharp increase in the response time around $x = 860$ flow instances is observed, which is exactly the same point where the performance of `SearchForPairs` rapidly degrades. This was previously attributed to lack of memory, but the detailed determination of the root cause and its underlying mechanics are left for future work.

Queries vs. rules As previously mentioned, the consequent part in a rule allows the ABox to be modified when the conditions in the antecedent are satisfied. The hypothesis to be tested was that modifications in the ABox might help simplify some aspects of ABox processing, for example by reducing the search space for subsequent queries. To test this hypothesis, the `ConsolidatePairsRule` previously defined (cf. page 150) was reused. The

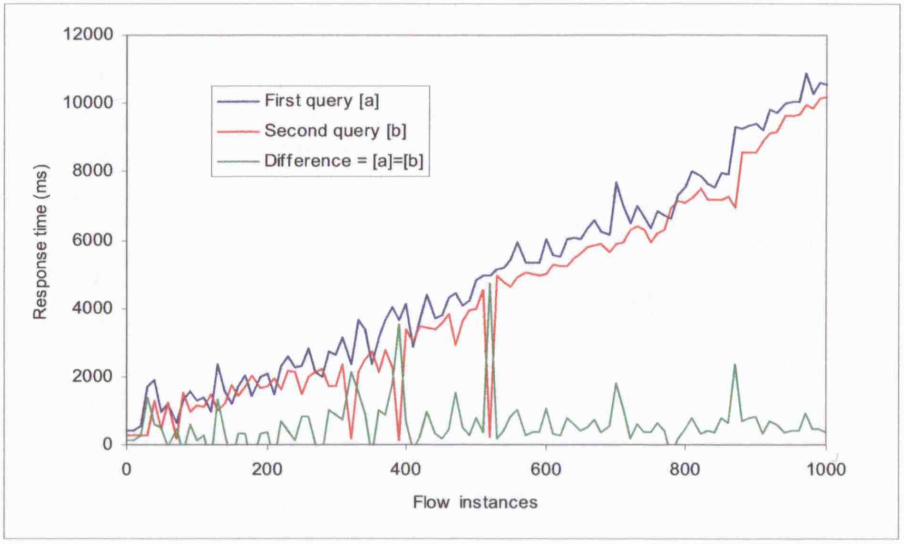


Figure 6.10: `ConsolidatePairsRule` response time

antecedent of this rule is almost identical to the `SearchForPairs` query, except for two additional atoms at the beginning that qualify the search space only to flows without any known counterparts, that is, they are not known to be related to other flows via the `hasCounterpartInverseFlow` relation. The rule consequent asserts this relationship when the antecedent conditions are satisfied. Note that `hasCounterpartInverseFlow` is not part of the assertions sent to the reasoner by the module that processes the packet traces; thus, the `NEG` operator with negation as failure (NAF) semantics is appropriate for this purpose.

Figure 6.10 plots the response time of `ConsolidatePairsRule` as a function of the number of flow instance assertions. Compared to `SearchForPairs`, its performance is generally better: at $x = 200$ flow instances, the response time is 2077 milliseconds (compared to 2868 for `SearchForPairs`); at $x = 800$ flows, the response time is 7552 milliseconds (compared to 8954 for `SearchForPairs`). While these still seem to be unacceptable for real-time flow processing, the performance improvement is quite noteworthy.

Two more observations may be drawn from Figure 6.10: first, the difference in response times between the successive rule "firings" is not as pronounced as in `SearchForPairs` (cf. Figure 6.8); and second, there is no sharp performance degradation (increase in response time) throughout the entire plot. These lead one to believe that the use of rules to modify the ABox may have either simplified the substrate structure, or reduced the memory requirements, or as previously hypothesized, reduced the search space for the query part of the antecedent.

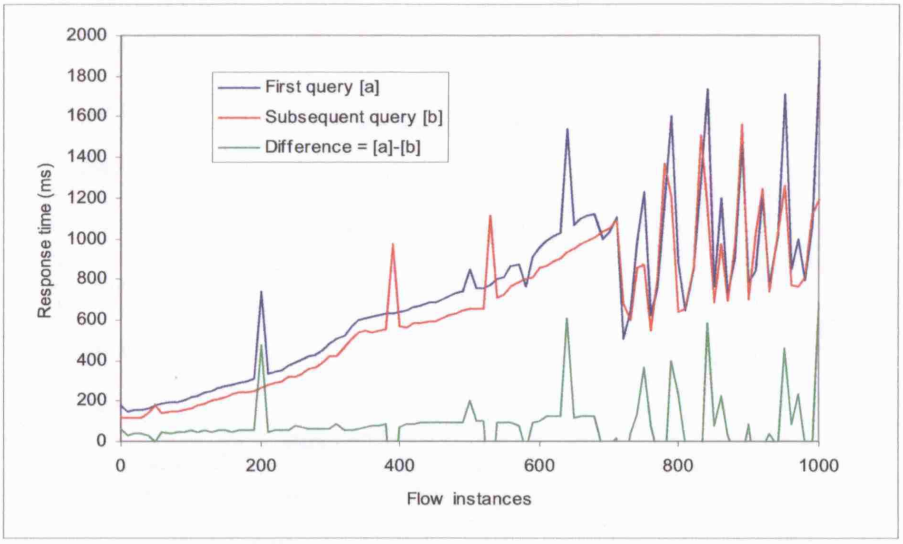


Figure 6.11: `SearchForPairs` response time using incomplete reasoning

Finally, a RacerPro-specific optimization called *told information querying*, which is one of the *incomplete reasoning modes* in nRQL, was explored. This mode uses only the information explicitly asserted in the ABox and relies only its relational structure for reasoning – in fact, it does not significantly use the TBox for reasoning [255]. The response time for the `SearchForPairs` query using this mode is shown in Figure 6.11.

Figures 6.11 and 6.12 show a more substantial improvement when incomplete reasoning is used, compared with the previous cases for queries that use complete reasoning and rule-based ABox modification. The first and subsequent queries in Figure 6.11 also track each other quite closely as in the `ConsolidatePairsRule` case in Figure 6.10. Except for the oscillations in the response time starting from approximately $x = 700$ flows, there is no singular sharp increase in response time anywhere within the graph. Although incomplete reasoning mode performs much better on average, it should be noted that it can only be applied for special cases where the scope of the query is limited to facts explicitly asserted in the ABox, rather than queries that would rely significantly on relations modeled in the TBox.

Summary

The results presented in the previous section show that the designer of a system that intends to use reasoner-based flow context aggregation would be faced with a number of design options and challenges, especially if the system is envisaged to operate in real time.

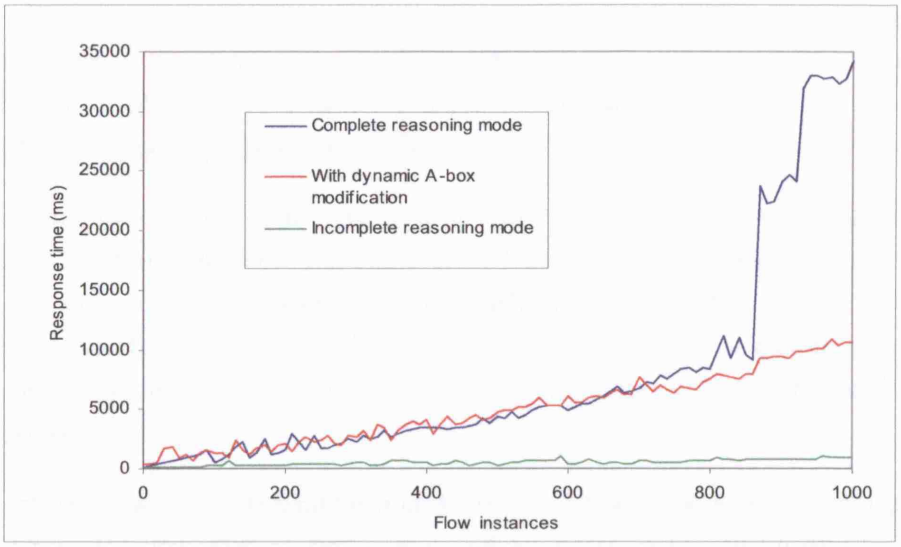


Figure 6.12: Comparative response times for queries using complete reasoning, ABox modification using rules, and incomplete reasoning

The results in the preceding chapter point to serious limitations in the total number of simultaneous flows that can be processed in real-time. This implies that in applications where real-time semantic aggregation of flow context is performed by a reasoner, there should be a small number of flows – implying applicability only to small networks, or to applications where the latency in reasoner-based aggregation might be acceptable within certain limits. An example of the latter might be in the case of adaptation to long-term flows, such as in video-on-demand or IP television. For short-term flows, the aggregation latency might be larger than a significant proportion of the flow lifetime (or it may exceed the entire flow lifetime), so reasoner-based aggregation might be applicable more for an offline analysis of the flow context information, rather than for real-time adaptation.

In the case of large networks, the reasoner should only deal with a small number of context tags, each describing a large bundle of flows (macroflows). These scenarios however are not too unrealistic: for example, large service providers that peer with each other may prefer to exchange a relatively small number of context tags, each describing large bundles of flows. As another example, a single Internet provider may distribute the aggregation process and push it near the subscriber access points, so that upstream aggregation would now deal with fewer, more high-level pieces of context information.

One obvious requirement would be the use of the appropriate hardware with sufficient processing power and memory. The limited capabilities of the hardware platform used for the experiments in this thesis certainly had a negative effect on the performance figures, and

thus places limits on the scenarios where real-time reasoner-based flow context aggregation may be used. Future work should study the performance of such a system using a platform with processing capabilities and memory capacity that might better reflect the class of hardware that may be deployed in real-life scenarios.

From a software standpoint, architectural decisions should be made regarding the division of processing tasks between sensors, other algorithmic pre-processors and aggregators (if any), and reasoners, in order to avoid performance bottlenecks, particularly in the reasoning component. In addition, the designer has to be aware of any platform-specific optimizations that may be available, and which of these, if any, would be appropriate for the context being processed.

Even as further advances in reasoning architectures and technologies are anticipated, the use of reasoners for flow context processing and aggregation has to be continuously and thoroughly evaluated. Some of the limitations of the methodology used in this section, as well as potential areas for its improvement, are described in Section 8.3.4.

6.4 Chapter summary

This chapter discussed the functions that serve as a bridge between sensing and using flow context: the discovery and location of context sources, the processing and aggregation of the information, and its subsequent dissemination within the network.

The first step in using context involves locating it: this thesis approaches that problem by equating it with the issue of locating flow context sensors. Through the use of a simulator, an approach to locating flow context sensors based on distributed hash tables was evaluated. The results of the evaluation seem to indicate that DHTs – and consequently many peer-to-peer network architectures based on DHTs [242] – may be used as a scalable, distributed mechanism for locating flow context sensors.

Once flow context sensors have been located, their information needs to be transmitted either directly to consumers, or to intermediate processors or aggregators. To serve as an aid in characterizing the approach in this thesis and classifying those presented by others, a generic framework for classifying dissemination schemes was developed. Two mechanisms for disseminating flow context, namely path-decoupled flow context dissemination, as well as a novel approach for context dissemination called path-coupled dissemination, were presented. Some approaches for their implementation were outlined, including a DHT-based approach for path-decoupled dissemination.

In some cases there is a need to process or aggregate flow context to transform it into something suitable for a consumer. The use of an automated reasoner for semantic processing and aggregation of flow context was discussed. The use of the flow context ontology in conjunction with reasoning actually demonstrates the dual uses of ontologies. While in Chapter 4 the process of ontology-building was used for *design-time*, abstract and conceptual flow context modeling, in this chapter the process artifact itself was used as a *runtime* component of a knowledge base supporting dynamic reasoning and semantic queries on flows and their associated context.

The experiments with semantic processing and aggregation presented here however showed that the process has significant computational costs. The results from the experiments with various platform-specific optimizations suggest that, in order to form generalizations about the use of reasoning-based semantic processing and aggregation, similar experiments be made using different reasoning platforms and hardware configurations. In addition, as will be explained in Section 8.3.4, a more realistic testing protocol, either involving a real-time replay of full packet traces, or testing on a live operational network, would provide a better testing environment.

Chapter 7

Using Flow Context: Applications

This chapter examines some application scenarios for flow context, particularly in mobility, quality of service (QoS), flow classification, in controlling or mitigating the effects of malicious or wasteful flows, and in network management. Except for the QoS application scenario, these applications were selected primarily to illustrate applications that had not been enabled by the early, QoS-centered prior work on network adaptation (cf. Section 2.2). The QoS application on the other hand is quite novel in the sense that it shows that the broad conceptualization of flow context allows QoS requirements to be signaled *implicitly* rather than explicitly.

These applications provide a sampling of the potential uses of context-awareness in networks, enabled by the flow context concept, in today's highly dynamic, heterogeneous, and mobile networks. The advent of user, host and network mobility for instance, especially in future wireless, ubiquitous and "ambient" networking scenarios (cf. Section 2.3.2 for example), requires networks to be able to deal with frequent changes in user location, connectivity, resources and activity as a result of her mobility.

On the other hand, the ability of service providers and network operators to identify and classify flows within their networks would allow them to cope better with the rapid emergence of new applications and the heterogeneity of existing ones. Additionally, as networks expand and more users become online, the number and diversity of threats to their operation correspondingly increase, pointing to a need not only to be able to identify and classify malicious or wasteful flows, but also to control or mitigate their effects. Finally, these new dynamic, heterogeneous and mobile networks ultimately have to be managed, either by a human or by machines: this section also shows some potential applications in this area, and discusses other potential applications in network management.

Before going into specific applications however, it is useful to examine common system issues in the design and implementation of networks that are "flow context-aware." These issues are presented in the next section, followed by the discussion on specific applications.

7.1 System issues

This section discusses some common issues that may be faced in the design, implementation and operation of a network that senses, distributes, aggregates or uses flow context. These issues are the result of an evaluation of the components and processes developed in the preceding chapters regarding the sensing, dissemination, aggregation and use of flow context. In addition, particular issues that arose during the implementation of the proof-of-concept application scenarios are mentioned where applicable.

7.1.1 Sensor deployment issues

The use of dedicated hardware infrastructures for sensing, such as in the case of many ubiquitous computing applications [12], in itself presents the problem of the associated costs of development or acquisition, operation, maintenance and future expansion. On the other hand, software sensors embedded into user and network devices may require a certain amount of overhead in terms of processing cycles and memory, depending on the nature of the context to be sensed, and the characteristics of the source of the context. For example, sensing the context of large numbers of high-speed flows might require specialized embedded software [177, 178], and in some cases, dedicated hardware support. Signature scanning, as described in Section 5.1.1, is often done in real-world scenarios by "boxes" dedicated for that purpose, such as in intrusion detection systems [55]. A designer thus has to decide whether the resources required by the planned sensing infrastructure – hardware or software – does not adversely impact the core functionality of the network and offers benefits that outweigh these costs.

Software-based flow context sensors (sources) may be deployed *statically* by providing either the necessary hardware infrastructure or by pre-deploying the necessary code or middleware. However, this often requires knowing in advance the types of context that would be used in a system, where they could be obtained, and the functionality needed to obtain it. In some cases this may be determined by the target application: for example, if an Internet service provider needs to obtain context information particularly about user flows (perhaps for traffic management purposes), then it would be logical to place sensors

at the point where subscribers access the ISP's network. On the other hand, if a company needs to protect its network from external attacks, the logical placement of the sensors would be at the gateways into the network. In a ubiquitous computing application where the types of flow context to be dealt with might be more diverse (e.g. including user device location), the sensing infrastructure might be more distributed and may involve devices outside the network itself.

An alternative would be the *dynamic*, on-demand deployment of (software) sensors within the system, using an active or programmable networking paradigm, as explored in Section 5.2.3. Although this offers a great deal of flexibility, there would be significant overhead involved in making the sensors nodes active or programmable, as they would need the necessary host platform for programmability (e.g. such as DINA in Section 5.2.3). For DINA in particular, there are also some practical limits on the level of flexibility that can be achieved: for example, access to low-level resources on the active node are provided by *brokers* that have to be programmed in advance. In this particular case, the programmability offered by DINA and the promise of flexibility is moderated somewhat by the need to write low-level resource brokers if the sensing function requires access to these resources.

Regardless of the method used in deploying sensors, once they are in place, these have to be brought on-line and made accessible to the other context components, such as context aggregators, as well the context consumers. Some of the methods available for resolving the location of sensors range from centralized directories or registries, to fully distributed approaches based on distributed hash tables (DHTs). As discussed in Section 6.1, there are trade-offs in terms of complexity and scalability when selecting any of these approaches. Typically, centralized approaches tend to be simple to implement but difficult to scale, while fully distributed approaches involve some complexity but scale quite well. The DHT-based distributed registry for flow context sensors evaluated in Section 6.1.1 offers low latency in resolving the location of context sensors and scales quite well; however, a full implementation along with the enhancements suggested in Section 8.3.2 would require additional computational and memory resources on each node participating in the DHT.

Another issue in locating sensors, which was not examined in this thesis, concerns the issue of searching for ranges or classes of sensors that might be able to provide a required type of context information. While it might be possible in some applications for context consumers to know in advance (i.e. in a pre-programmed or pre-configured way) the exact types of sensors needed for a certain task, in other cases a less exact, "closest-match" type of search (for some given criteria for similarity) might be useful. This was not examined in this thesis and is suggested for future work.

Finally, a standard interface and protocol is needed in order to communicate with context sensors. In the experiments in this section, the sensors used a simplified implementation of the ContextWare primitives mentioned in Section 6.1.1, developed by others. A full discussion of the implementation of these primitives is beyond the scope of this thesis; more information may be found in [109, 233].

7.1.2 Flow context dissemination

The dissemination of flow context tags within the network also consumes bandwidth in itself, so another issue concerns striking a balance between the network overhead and the expected level of optimization or benefit that the use of the information (e.g. for adaptation) can provide. A flow context tag may contain a very brief and simple descriptor for the associated flow, such as its QoS *flowspec* [142], or the coordinates of the sending host, or perhaps a specification of the media encoding of the payload. Conversely, it would also be possible to construct a rather complex description of the flow that includes descriptions about the user's activities, the capabilities of the sending or receiving hosts, the QoS description of the flow and its associated payloads, and so on.

Determining the best mode of disseminating flow context is another system issue that designers must address. Section 6.2.2 describes path-decoupled and path-coupled approaches in dissemination, and discusses their advantages and disadvantages. The method developed and used in this thesis, the path-coupled approach, disseminates flow context along the flow's path, where it is *likely* to be needed. However, interested consumers that are not directly along the flow path would probably need special mechanisms to ensure that they are also able to receive such context information, perhaps by sending out notifications to neighbors that they are interested parties (this was not investigated in this thesis). Another potential issue with path-coupled dissemination concerns the ability to ensure that the context tag stays "coupled" with the flow, that is, that the tags and the flow in fact traverse the same path through the network. In the case of the experimental work done in the next few sections, the tags were transmitted as a separate UDP stream, so there were no guarantees that the tags would not be misrouted or dropped within the network.

Another issue not addressed in this thesis concerns maintaining the integrity of the transmitted flow context from a security viewpoint. This is suggested for future work.

7.1.3 Context aggregation

A major issue that a system designer must face concerns the performance of reasoner-based semantic aggregation, discussed in Section 6.3.2. As shown in that section, for the specific hardware platform used in experimentation (with an Intel Pentium M 1.7GHz processor and 1GB memory), the performance of reasoner-based aggregation was quite poor and unsuitable for applications that required real-time semantic aggregation of large numbers of flow context tags. In that section a recommendation was made to either use that mode only in small networks, or to ensure that the number of tags to be aggregated at each stage would be relatively low, perhaps by employing a multi-stage approach to aggregation. For example, a certain amount of aggregation might be performed at the edge, followed by successive aggregation towards the core.

It should be pointed out that there are other possible modes of aggregation that do not involve reasoning. As enumerated in Section 6.3, aggregation may also be in the form of data formatting, range checking and data validation, algorithmic computation and normalization, data fusion, and augmentation. Thus, the performance impact of flow context aggregation is determined by the complexity of the aggregation required. In some cases, as in the proof-of-concept experiments done later in this chapter, simple forms of aggregation were employed (and required, because the examples were purposely kept simple), so the the computational and storage overheads, as encountered in these experiments, were minimal.

The requirements for storage and state information associated with flow context tags is another crucial issue. The potential issues with state and storage are similar to the concerns raised with the RSVP-IntServ model, which provides QoS to individual flows through per-flow queuing and scheduling in the data plane, and per-flow reservation state management in the control plane [142]. However, it has been pointed out that much of the negative impact on RSVP-capable routers stems from the per-flow queuing and scheduling (the per-flow data plane operations), since the per-flow reservation messaging overheads are somewhat controllable [256]. Analogously, it could said that the state and storage impact of the use of flow context is an application-specific issue that can be subdivided into two parts: the number and size of flow context tags, and the resulting state based on the control-plane actions taken by the node in response to this context. As discussed in Section 6.3.2, the amount of context state that needs to be stored depends on the amount and type of context information required by the context-"interested" node, the granularity of the information (i.e., whether the context describes individual flows or bundles of flows), and the lifetimes of the flows of interest (whether these are short-lived or long-duration flows). For example, consider an Internet service provider that wishes only to collect per-user flow context near access nodes and concentrators, such as per-user usage statistics.

The storage requirements for this simple example would probably not exceed the typical storage requirements of most billing and traffic monitoring systems in current Internet service providers. At the other extreme, a network where all nodes are context-aware and which collect and process flow context in a fine-grained way would certainly feel the impact of the storage and state requirements of flow context.

The aggregation of flow context information may also be a potential way of keeping the amount of flow context (and thus context state) in a network within reasonable limits. As mentioned in Section 6.3.1, aggregation may be done across flows, across time, or across different sensors. The aggregation of flow context across flows would effectively decrease the granularity of the flow context descriptions, and thus require less state for the same total number of flows. Some types of temporal aggregation on the other hand could at least prevent the amount of state per flow from increasing with time: for example, storing the moving average of the current flow rate would take up significantly less space than, say, storing the instantaneous flow rates gathered at regular intervals.

One technique employed in RSVP is the use of *soft state* in order to maintain reservation state in routers [142]. Reservations disappear by themselves if not refreshed periodically, avoiding orphan reservations and allowing quick response to routing changes [257]. Although not explored in this thesis, a similar technique might be applied to flow context stored in nodes: if not refreshed, these could be aged out. In addition, a quality of context (QoC) parameter associated with the tag may also be used to describe its validity; beyond the specified amount of time, the context information expires, freeing up storage space.

7.1.4 Operational aspects of the ontology

Once the ontology has been set up, it can be used in a variety of ways as described in Section 4.2. These use patterns may either be "offline" design-time uses for the ontology, as well as "online" runtime uses. Some examples of design-time uses include the use of ontologies as common implementation-neutral languages for the authoring and translation of domain artifacts (such as contextual descriptions), and as specifications for modeling and requirements analysis during the design and implementation of systems and their components [125]. Runtime uses for the ontology might include the use of ontologies as vocabularies for online *translators* between network or application domains that use different representations or implementations for objects [125], or as input to reasoners for inference and semantic aggregation of context (cf. Section 6.3.2).

To ensure broad acceptance and use, ontologies need to be stored and made available to

other domain experts and users. Publication of the ontology (such as through the World Wide Web) encourages collaborative review, revision, adoption and extension by wider communities. In this regard, standardization, such as through the efforts of the Foundation for Intelligent Physical Agents (FIPA) as mentioned in Section 4.4, is invaluable.

From a more operational viewpoint, when not directly embedded into software components or stored locally, entire ontologies might be made available within and across domains through shared ontology services. Ontology servers might serve as ontology repositories, accessed in behalf of clients by ontology agents [167]. As likewise mentioned in Section 4.4, FIPA even provides a specification, including an ontology, for ontology services that provide ontology access and related services to agent communities.

The size and complexity of the ontology may be an important issue in determining its potential application area, particularly if it is to be used in a runtime fashion. For example, the results in Section 6.3.2 suggest that the slow performance of reasoner-based flow context aggregation make it more suitable either for small networks, or perhaps in larger networks where the pre-processing and hierarchical aggregation of context results in fewer context tags to be processed, or in applications where the flows are long-lived, such as in video streaming applications. Conversely, it is possible (although it has not been exhaustively verified in this thesis) that for applications where flows tend to be short-lived, the ontologies should be smaller or less complex, or perhaps only a relevant subset of a larger ontology must be used. This is left for future investigation.

As the ontology is progressively used, it is expected to evolve, either as a result of its continuing validation, or in order to deal with changes in the underlying domain, or to cope with heterogeneities. The evolutionary process may involve updates and modifications, as well as the integration and reuse of other ontologies. Section 4.3.2 mentions several approaches to ontology integration and reuse, including ontology extension, translation, and determining equivalences [145, 146]. Other approaches to integration might include polymorphic refinement (where a definition is extended and included so that it can be used with other arguments), restriction (simplifying assumptions are made that restrict the included axioms), or a merger of commonalities [258]. From a systems point of view, in the case of runtime ontology services, there should be management mechanisms by which the shared online ontologies – which might even be distributed in multiple servers – are likewise updated to reflect these changes. These management mechanisms should include provisions for notifying ontology clients so that they may either download new versions or incrementally update their current ones if feasible.

7.1.5 Adaptation

Finally, some system issues that need to be considered when using flow context to perform adaptation within the network also include their performance and resource impact on network nodes, the location and deployment of adaptor components.

Locating even the most basic adaptor components within network nodes should be carefully considered, as these may consume significant amounts of resources and negatively impact the basic forwarding performance. As shown in studies with RSVP, a significant amount of overhead was contributed by the per-flow QoS treatment rather than the RSVP signaling overhead. Similarly, it may be possible for the per-flow *adaptation* (QoS-related or otherwise) to consume significant amounts of resources, rather than the flow context tags per se. Again, this means that the amount of overhead would be application-specific, that is, it would depend on the type of adaptation required on each node, as demanded by the application in mind. In the experiments in this chapter, many of the adaptation mechanisms used the traffic control techniques in Linux [144]; again, as the intention was to demonstrate proof-of-concept, the examples were kept simple. How these examples would perform under conditions of increasing loads was not tested, and is left for future work.

One way of dealing with the impact of adaptor overhead would be to offload the adaptation process to an *overlay*. This would ensure that flows that would need to be adapted would be diverted away from the default "fast path" and that the adaptor components would consume resources independently of the forwarding nodes. The use of separate adaptor overlays, however, was not investigated in this thesis.

Finally, in contrast with the static deployment of adaptor components (such as in dedicated, pre-configured overlays), it is also possible to dynamically deploy adaptors using active and programmable networking approaches. The issues involved in this case are similar to those discussed in Section 7.1.1 concerning the use of active networks in dynamically deploying sensing functionality.

The next few sections now discuss some potential application scenarios for flow context.

7.2 Mobility and moving networks

This section explores the use of context tags containing location and mobility information in aiding the handoff process, by enabling networks to institute proactive measures in anticipation of handovers, rather than react after a handover has taken place. A simple

proof-of-concept experiment that demonstrates its usefulness in handing over real-time UDP streams, such as audio and video streams, is presented.

7.2.1 Background

In Mobile IP [259], packets sent by a corresponding node (CN) to a mobile node (MN) are routed first to the MN's home network using the MN's permanent home address. At the home network, the mobile node's home agent (HA) intercepts such packets and tunnels them to the mobile node's most recently reported care-of address (COA), the temporary address assigned to it on the foreign subnet. At the endpoint of the tunnel, the inner packets are decapsulated and delivered to the MN. In the reverse direction, packets sourced by mobile nodes are routed to their destination using standard IP routing mechanisms.

The fact that packets have to be sent first to the HA and tunneled to the FA is called the *triangle routing problem*, and raises concerns that this causes packets to traverse paths that are longer than optimal. A technique called *route optimization* [260] enables CNs to receive binding updates containing the MN's COA, so that the CN can tunnel packets directly to the MN without passing through the MN's HA. However, even with route optimization, the handoff process may introduce sufficient latency and packet loss as to adversely affect real-time flows, such as audio and video flows.

7.2.2 Approach

Previous work by Stemm and Katz [261] showed that handoff latency is dominated by the *discovery time*, the amount of time before a mobile discovers that it has moved into or out of a new wireless overlay. This experiment attempts to minimize the effects of the discovery time through a technique called *speculative multicast*. In this technique, multimedia streams are multicast to base stations where the mobile node is likely to be handed off. Figure 7.1 shows a mobile node MN requesting a multimedia UDP stream from server CN. The stream is sent by CN to MN, the latter accessing the network via access point AP_A. MN then moves from the coverage area A of AP_A to coverage area B of AP_B. As the MN moves, it provides updates of its location through context tags. The mobile node may obtain its location using any number of means, including the Global Positioning System (GPS) [222] outdoors, or indoors using techniques such as RADAR [205] and its variants. Other techniques [206, 207, 221], including the one developed in this thesis (discussed in Section 5.3 on page 112) may provide more high-resolution indoor location estimates, accurate to within 10–20 centimeters.

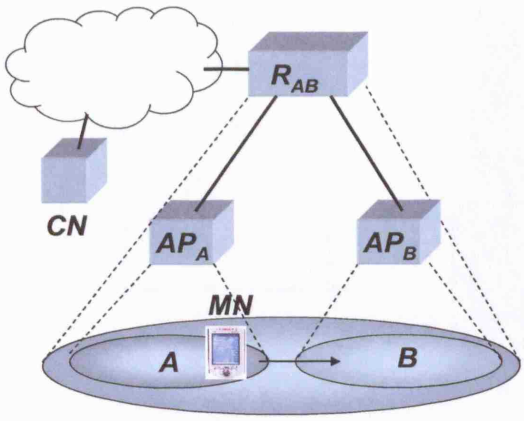
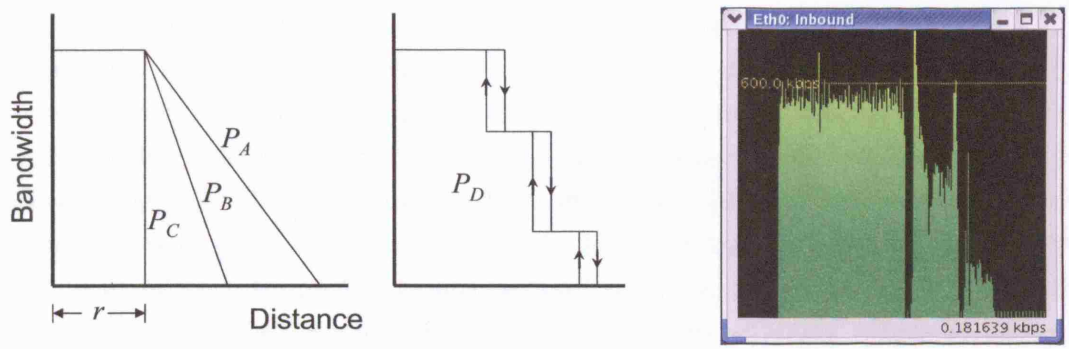


Figure 7.1: Network diagram for mobility experiment

As MN moves into coverage area B, router R_{AB} detects through the MN's context tags that it is starting to move within proximity of the coverage area of AP_B. It then starts to multicast the return stream to AP_B. While MN has not yet formally handed over to AP_B, is up to AP_B whether the received speculative multicast packets will actually be transmitted over its radio interface, or simply cached. When MN has formally been handed off, the new AP notifies the upstream router.

The context tag within the return stream also indicates the type and characteristics of the flow content, and whether it can be further distilled [20] through lossy compression or transcoding, or by dropping layers [262, 263, 264, 265] in order to reduce its bitrate. If the bitrate of the inbound stream can be modified, it is reduced according to an *adaptation profile* such as the one shown in Figure 7.2a. Conceptually, the current distance of the mobile node from the prospective next access point serves as a rough measure of the likelihood that it will indeed be handed off to that access point: the closer it gets to the access point, the higher the probability that it will in fact join that access point. The adaptation profile therefore reflects this by speculatively multicasting in proportion to this probability. When the likelihood of joining an access point is small, the bitrate of the stream sent to the access point is proportionately small. The idea behind this is to minimize the *miss-penalty*, or the number of bits speculatively multicast to an access point that are not actually received by the mobile host because it has not joined the access point. It is up to the access point whether streams received via speculative multicast are actually sent over the radio interface, or simply cached within the access point.

The adaptation profile may also reflect whether the network is adopting either a conservative or aggressive policy towards speculative multicast. In Figure 7.2a the profile with the steeper slope P_B reflects a more conservative policy than P_A, as it allows a high bit



(a) (b)

Figure 7.2: (a) Adaptation profiles. (b) Traffic response on router.

rate for the multicast stream only when a mobile node has a high probability of joining the access point. An obvious advantage of a conservative policy is that it has a lower expected miss-penalty. One potential disadvantage is that it requires the router to be agile enough to respond to a mobile node traveling at high speed. The degenerate case would be to have an adaptation profile P_C that corresponds to a policy that only allows streams to actually be sent to a mobile node that is actually within the coverage area of an access point.

While P_A and P_B represent somewhat idealized adaptation profiles that exhibit linear variations in the bit rate as a function of distance, P_D represents a more realistic profile that implements a stepwise adaptation. An adaptation profile may also be dynamically modified by the router, primarily for two reasons: first, to implement hysteresis, as shown in P_D, thereby preventing oscillations in the bitrate as a result of noise in the location estimates or small actual changes in location; and second, to account for resource availability within an access point. In instances where there are many MNs that may be potentially sharing a particular access point, the router may shift to more conservative profiles in order to assure the availability of resources such as memory in the case of caching, or bandwidth if access points transmit speculatively multicast packets over the radio interface.

7.2.3 Proof-of-concept experiment

To demonstrate the use of context tags in mobility, a simple proof-of-concept experiment was conducted. The movement of a mobile node receiving a video stream was simulated by injecting context tags that specified the position of the mobile node. The mobile node started from the center of an access point/router's wireless coverage area, moving radially away in a linear fashion, at a constant and relatively slow rate. The router used a three-step adaptation profile similar to P_D in Figure 7.2a, without hysteresis.

Figure 7.2b shows the inbound bandwidth on the access point servicing the mobile node. The maximum bandwidth received by the access point is when the mobile node is within the coverage area. Even as the mobile node leaves the coverage area, the access point still receives the video stream from the router, although the bandwidth has been reduced by transcoding. The discontinuities in the bandwidth response (narrow areas where the bandwidth dips to zero) of the router are the result of the transcoder restarting when it goes to a different target bandwidth setting; these discontinuities can be minimized by either improving the response of the transcoder or by adopting a layered instead of distillation-based approach to video stream adaptation.

7.2.4 Related work

One mechanism proposed by Stemm and Katz [261] to reduce handoff latency is the use of *doublecasting*, whereby packets within a specific wireless overlay are also simultaneously sent to another base station belonging to the next higher overlay in the network's hierarchy. This technique however assumes that the mobile host simultaneously receives from the two overlays on *different* network interfaces, and is thus applicable for what they called *vertical handoffs* between overlays that use different network technologies.

Helmy, Jaseemuddin and Bhaskara [266] proposed *multicast-based mobility* (M&M) as a mechanism to improve handoff performance. Within a domain, a mobile node is assigned a multicast address and throughout its movement, joins this multicast address through locations it visits. Correspondent nodes wishing to send to the MN send to this multicast address. Due to geographic proximity, it is likely that the join from a new location will be a few hops away from an already-established multicast distribution tree. In their approach, the mobile host (re-)joins the multicast tree only after it has actually moved to the new location. This is different from the approach in this thesis, where the inbound stream may actually be sent to the base station or access point covering the *possible* next location for the mobile host, even before it has actually moved there.

7.2.5 Summary

This proof-of-concept scenario illustrates the interplay between two different types of flow context: node location, which is extrinsic, and content type, which is intrinsic. The adaptation proposed would not have been possible without having both types of context available.

The technique described is geared towards environments and applications where handover

performance is more critical than bandwidth efficiency, thereby making speculative multicast a viable proposition. For example, in a residential network, one might have relatively few users sharing wireless access points that deliver modest-to-high amounts of bandwidth, with a fairly low utilization rate. Another environment where this scenario may be applicable might be one where you have a relatively large number of *picocells*, each providing a large amount of bandwidth over a small area, with a low density of mobile devices at any given instant. In this case, users would tend to experience a larger frequency of handoffs as they roam, since they would tend to move quickly from the small geographical coverage of one picocell to the next. Furthermore, conservative adaptation profiles may be used in situations where bandwidth efficiency is also a consideration, thus making speculative multicast a general mechanism that may be considered for most mobile environments.

Flow context tags may also be used in conjunction with better mobility prediction techniques; the use of distance as a predictor in this experiment is merely illustrative rather than prescriptive in nature. Adaptation profiles could use a more general "join probability" rather than distance as a dependent variable in determining the multicast bitrate.

7.3 Implicit QoS signaling

Mechanisms for providing Quality of Service in networks described in the literature often expect end-hosts to either explicitly signal their QoS requirements and undertake resource reservation, or for them to have sufficient knowledge about the underlying QoS model in order to map application flows to existing QoS classes. For example, signaling messages in the Resource Reservation Protocol (RSVP) [142] and the draft IPFIX model [147] contain traffic descriptors such as bandwidth or token bucket parameters, QoS class parameters such as the Differentiated Services (DiffServ) code point (DSCP) [199], and other QoS characteristics such as transfer delay, delay variation, packet loss and bit error rate. While these parameters indeed provide a detailed description of QoS characteristics, they may not be sufficient to paint a "big picture" that better describes the desired interaction between the user and the network.

This section explores the use of context tags in implicitly signaling the QoS characteristics and requirements of network flows. The mechanisms are first discussed, then demonstrated through a simple experimental scenario. Similar efforts in this application area are then discussed.

7.3.1 Mechanisms

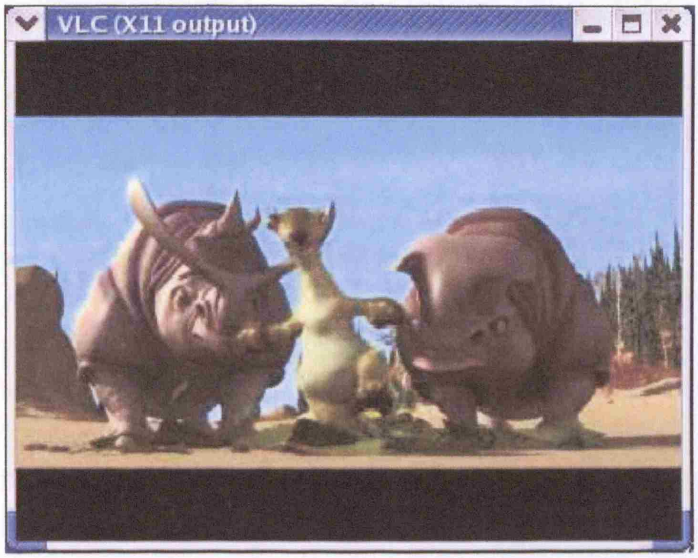
Context tags may be used for implicit QoS signaling in any of the following ways:

- To decouple end-hosts and applications from the underlying domain-specific QoS model. In this case, it would be up to the network to derive the QoS requirements from a high-level contextual description of the flow, and to map them to the specific QoS mechanics within the domain. This is also useful as a mechanism for assuring QoS on a flow as it traverses two domains with different QoS implementations, such as when two domains use different DiffServ Codepoint-to-Per Domain Behavior (DSCP-to-PDB) mappings [199, 267]; or between heterogeneous frameworks, such as between a domain that uses Diffserv and another that uses Multiprotocol Label Switching (MPLS) [143].
- To provide or expose additional information about the flow to the network in an explicit way to facilitate flow classification for QoS purposes. For example, information about the multimedia capabilities of a mobile terminal may be embedded within the application flow. This information would not be normally available to network devices without computationally expensive stateful application-layer flow inspection. Context sensing functionality could instead be deployed within end-hosts or dedicated network middleboxes could perform flow inspection or extract context from higher layers and share this information via context tags to multiple adaptors downstream.
- To trigger QoS adaptation on the flow. For example, the content of a flow may be compressed or transcoded in response to QoS constraints, device capabilities, user activity or application requirements.
- To tag and help identify suspicious and malicious flows, or those that are in violation of QoS contracts, in order to keep the network within engineered QoS limits as much as possible

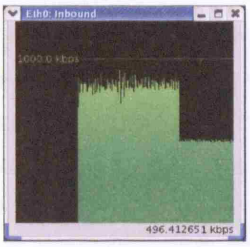
7.3.2 Simple proof of concept

The following provides a simple demonstration on the use of context tags to implicitly signal the QoS characteristics and requirements of network flows.

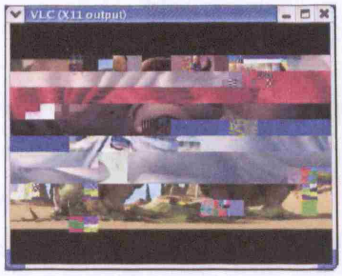
A video stream with a natural bit rate of approximately 850 kbps was transmitted over the network, resulting in video with typical quality shown in Figure 7.3a. Assume that for some



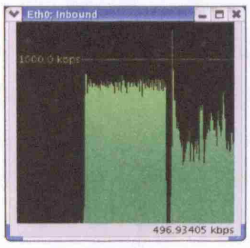
(a) Image captured from original video with no bandwidth restrictions



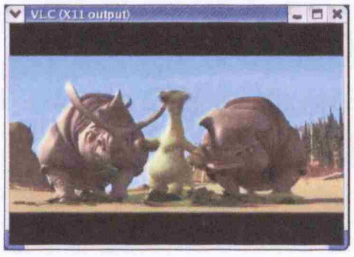
(b) Bandwidth limiting



(c) Resulting video stream without transcoding

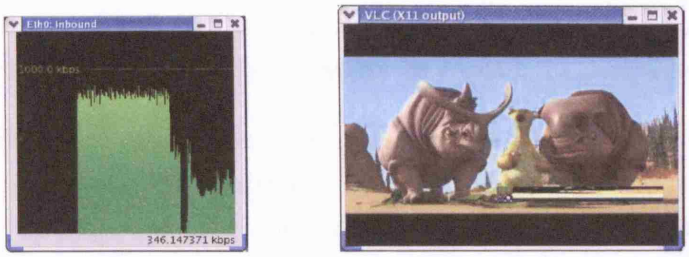


(d) Adaptation by transcoding



(e) Resulting video stream with transcoding

Figure 7.3: QoS adaptation on untagged (b,c) and context-tagged (d,e) video streams



(a) (b)

Figure 7.4: QoS adaptation using a combination of transcoding and rate-limiting strategies. (a) Traffic profile. (b) Resulting video quality.

reason, the network has to reduce the average bitrate of this flow to a target 500 kbps, for example to execute an adaptation profile similar to those discussed in Section 7.2, as shown in Figure 7.2a. A simple (and perhaps default) way for this network to achieve this would be to impose a hard limit on the allowable bit rate of this flow, as shown in Figure 7.3b. While this would be suitable for elastic flows (i.e., flows that can adapt to changes in traffic characteristics such as available bandwidth and delay), this can be unsuitable for packet loss- or delay-sensitive traffic such as video. Figure 7.3c shows the qualitative effect of context-unaware QoS adaptation on the video stream resulting from the absence of the appropriate flow context. The packet losses and delays resulting from the bandwidth hard limiting degrades the video quality to an extent that may be annoying to users. On the other hand, injecting the appropriate context tags into the flow informs the downstream adaptor that the flow contains video whose current encoding allows further transcoding. The adaptor uses this information to trigger a transcoding function, setting the output bitrate parameter corresponding to the 500 kbps target traffic rate. The traffic profile of this adaptation strategy and the corresponding video quality are shown in Figs. 7.3d and 7.3e, respectively. The traffic profile shows some "spikes" which are artifacts of the transcoding scheme used. In order to prevent these, a combination of the bandwidth limiting and transcoding adaptation strategies were used, resulting in the traffic profile and video quality shown in Figs. 7.4a and 7.4b, respectively. Occasional and minor degradation of video quality was observed in this case, but the overall quality experienced was subjectively acceptable.

Figure 7.4 also demonstrates *adaptor composition*, where a complex adaptation function is built (composed) from more basic adaptation modules. It is likely, although not demonstrated here, that the function would not be realized properly if band-limiting the flow preceded transcoding its content. This problem may be handled by explicitly stating the

interoperability parameters of each adaptor (or service module as a generalization), specifying the input and output conditions necessary to cascade or compose them [23].

This simple experiment shows how context tags may be used to signal QoS requirements and acceptable QoS adaptation strategies. The example demonstrates *implicit signaling*, as the sending host had no prior knowledge of the QoS adaptation model existing within the network. In this case it was up to the network to decide which adaptation strategies were appropriate, given the flow's context and the network's QoS goals.

7.3.3 Related work

HQML is an XML-based hierarchical QoS markup language targeted for World Wide Web applications [101]. One of its useful features is that it allows applications to signal QoS characteristics and requirements not only to end-applications, but also to network elements called QoS proxies. However, it is focused specifically on QoS and on Web applications, and is not designed as a general mechanism for making other types of context information available to network nodes.

The Session Description Protocol (SDP) [94] describes multimedia sessions using a short textual description that includes information on media, protocols, codec formats, timing and transport information, while Multipurpose Internet Mail Extensions (MIME) [21] provide high-level type descriptions for different content types such as text, images, video, audio or application-specific data in message streams. Unlike context tags, these schemes deliver flow or session context to end-hosts rather than network nodes, and are limited to very specific application domains. However, the formats and types used in SDP and MIME messages may be used to describe flows in a high-level way within context tags; these could have also been used in the example application to signal an adaptor that transcoding would be a viable QoS adaptation strategy.

7.3.4 Summary

The ability to provide QoS to flows will be an important feature of future context-aware and adaptive networks.This section demonstrated, using a simple experimental scenario, how context tags may implicitly signal QoS requirements. In implicit signaling, flows express their QoS requirements implicitly and trigger QoS adaptation services without being aware of the details of the network's QoS mechanisms. Proxies or other devices within the network may use this context information and perform explicit QoS signaling in

behalf of the flow (and end-host). Alternatively, network elements themselves may perform the necessary QoS adaptation based directly on the flow's context, without the need for low-level QoS signaling.

7.4 Intelligent flow classification and management

There are instances where information needed for flow classification such as network addresses or transport-layer port numbers are modified, such as when network- or port address translation (NAT/PAT) are used, or when traffic is tunneled within well-known protocols such as the Hypertext Transfer Protocol (HTTP) as a stealth technique or as a means to bypass firewalls [64, 183, 184, 215]. Although it is possible to obtain information on the nature of the application generating the flow and its high-level content either through stateful inspection on the application-layer payload of each packet, or by application-layer flow termination, this would undoubtedly be computationally expensive and impact the performance of network nodes such as routers, if *each* downstream node had to perform its own stateful inspection. As an alternative, the flow's context may be sensed either at end-hosts or by dedicated sensors, such as `FlowSensor` (cf. Section 5.1 on page 99), and this information could be shared throughout the flow path, so that the network may properly classify the flow.

This section presents a simple experiment that demonstrates the use of context tags in aiding flow classification. It shows that context tags allow flows that would normally appear identical to the network to be further classified and given differential treatment within the network when necessary.

7.4.1 Proof of concept experiment

This experiment assumes a scenario where a user wishes to view two video streams simultaneously over a link with limited bandwidth. Both streams are encapsulated in UDP and use the same type of video encoding, so for all intents and purposes they are virtually identical from the network's point of view. They only differ in terms of their content, and the subjective notion of importance applied by the user on that particular type of content, based on the user's current activity. For example, if the user is relaxing at home, then she might consider work-related video content to be of secondary importance compared to entertainment-related video content, and vice-versa. In this experiment these user activities are not sensed, but set manually (i.e. simulated). Additionally, the subjective priority

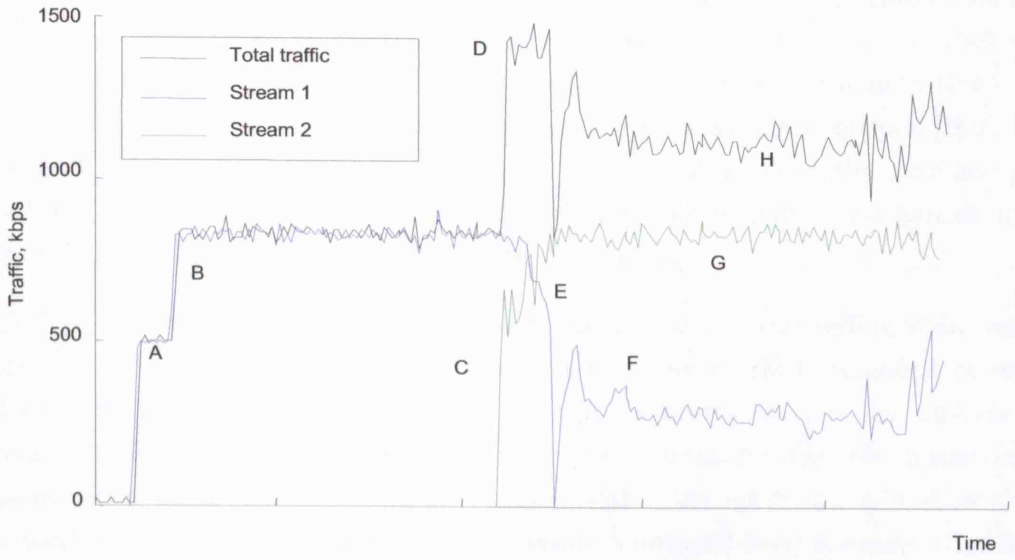
or importance level assigned to each video stream are likewise manually assigned.

The experiment was started by requesting a video stream over a link initially allocated 500 kbps bandwidth. This video stream had a natural (uncompressed) rate of approximately 850 kbps, and the network was configured to initially adapt by allocating at least 1.5 Mbps to the link whenever video was transmitted over it. The user then requests an additional copy of the same video stream, however, the context tag of this stream gives it higher priority, i.e., it is subjectively the "more important" stream as far as the user is concerned, at that point in time. Both video streams were fairly constant bit-rate (CBR) streams in their uncompressed state, so the statistical sum of their offered traffic rates would, on average, exceeded the 1.5 Mbps bandwidth allocated to the link. The network's adaptation policy was configured to allow the flow with the "high-priority" tag to be transmitted without constraint, while the lower priority flow was adapted by transcoding it down to a lower bit rate. This kept the total traffic within the 1.5 Mbps limit.

Figure 7.5 shows the result of this simple experiment. The user, initially allocated 500 kbps, views the video stream as shown in point (A) on the graph. The context tag within the stream results in a new bandwidth allocation of 1.5 Mbps, allowing the video stream to rise to its characteristic level of around 850 kbps as shown in (B). The user requests the additional video stream, and commences viewing the stream in (C). The combined traffic saturates the bandwidth allocation as shown in (D), resulting in degraded video for both streams. The network receives the new context tag for the higher-priority video flow and aggregates this with the context tag of the lower-priority flow, allowing the streams to be prioritized with respect to each other in (E). Although the flows have now been prioritized, the total offered traffic still exceeds the allocated 1.5 Mbps bandwidth, so the network further adapts by applying transcoding on the lower-priority stream. The lower priority stream now operates at a lower average bitrate after transcoding (F), while the higher priority stream occupies its natural traffic level (G). At this point the total bandwidth consumed stays well within the 1.5 Mbps allocation as shown in (H).

7.4.2 Summary

Flow context tags may be used to provide information that may aid the classification, and if necessary, differential treatment or adaptation of flows within networks. This is especially useful in encoding information that cannot be expressed in packet headers, or that cannot be reliably or economically obtained through packet or flow inspection. The scenario presented demonstrated an extreme case, where two *identical* flows were classified and given differential, context-aware treatment by the network.



(a) Graph showing bit rates of both inbound video streams

(b) High-priority stream (uncompressed)

(c) Low-priority stream (compressed)

Figure 7.5: Network adaptation using context tags for flow classification

7.5 Mitigating attacks and controlling malicious flows

Equipped with sensors that can detect distributed denial of service (DDoS) attacks or worms propagating through the network (e.g. [268]), a node may inject a context tag in the reverse path so that upstream nodes may suppress the inbound flows. The tag may contain a general description (filterspec) of the malicious flow that upstream nodes may use as a pattern to detect and suppress subsequent attacks, ultimately at or near their sources. This technique would have better prospects of working across administrative domains, rather than low-level traffic suppression mechanisms such as ICMP quench [269], because (a) context tags are designed to be interpreted by the routers along the path and not just by end-hosts, and (b) more detailed information (possibly readable by a human operator) is provided regarding the request to suppress the traffic.

Similarly, this technique may also be explored as a means of controlling spam email and mitigating its impact on the network. Spam is typically marked, classified or discarded at the receiving end; by the time it has reached its destination, it has already wasted a portion of the network's bandwidth and has considerably taxed the resources of the recipient server [270]. Context sensors deployed within the network could allow the early, in-network detection of spam and possibly enable a network-level response that may also propagate all the way near the source of the spam traffic. This is especially significant in suppressing spam traffic that may be originating at a high rate from hosts hijacked by mass-mailing malware.

Network-level responses to spam would involve some form of blocking, rate-limiting, or rerouting. Blocking would be an extreme form of response where a network refuses to transport traffic previously sensed and classified as spam. However, this may be rather dangerous, given that there is always a certain level of inaccuracy in sensing and classifying spam [54]. Rate-limiting schemes, such as teergrubing [271], involve slowing down the SMTP conversation in order to limit the rate of the inbound spam, thereby taxing the resources of the spammer as well. Li et al. on the other hand suggested *TCP damping*, where a recipient makes a TCP connection appear inefficient to a suspected spam sender either by delaying acknowledgments, closing or minimizing the advertised window, or faking congestion [272]. While these are typically done by the recipient SMTP server, a network-level response could mimic these by dropping selected packets from the spam flow, or replicating acknowledgments, or some other means, thereby inducing TCP congestion avoidance mechanisms [273, 274] (which however may unfortunately also cause retransmission and waste more bandwidth), or better yet, by dropping or delaying selected application-level messages. Another form of network-level rate-limiting would involve bandwidth-limiting the

suspected spam flow. Finally, rerouting the network flow to less-critical or less-expensive paths or links would allow the network provider to mitigate its effects on the network without unduly penalizing the intended recipient in the event that the email flow was misclassified as spam.

Flow context can be used as a means of promoting a cooperative approach to mitigating malicious or unwanted flows. Clayton for example proposed *extrusion detection*, that is, the examination of *outbound* flows, rather than the usual examination of inbound flows, to detect spam [275]. Although it might not be feasible for a provider to block the outbound transmission of suspected spam due to regulatory or contractual issues [275], it might be feasible for a provider to tag such outbound flows with the appropriate contextual description so that a peer cooperating provider could apply an appropriate response, such as those outlined earlier. The tagging of outbound flows may also be considered for classes of suspicious traffic other than spam. Finally, in some extreme cases where a provider internally verifies that an outbound flow has a malicious (rather than just suspicious) nature, it may decide to entirely block its transmission. While this might involve the manual examination, decision-making and intervention by a human operator, the mechanism that could alert the decision-maker in the first place could be flow context.

7.6 Network management

Flow context may be used in network management and engineering is another area. For example, the flow visualization tool presented in Section 5.1.2 presents an interface similar to the popular open-source Multi Router Traffic Grapher (MRTG)[1] application used for network traffic monitoring and management. The main difference between the flow visualization tool presented in this thesis and in MRTG is that in the former, the traffic graphs are on a per-flow basis; in the latter, the traffic graphs are typically for router interfaces.[2]

From a wider management perspective, such as from the "classical" FCAPS management framework – fault, configuration, accounting, performance and security – it is also possible to envisage applications of flow context.

In fault management, flow context sensors detecting PDU errors or losses beyond a certain threshold, or the unexpected (from a stateful point of view) disappearance of a flow, might trigger an immediate adaptation within the network to reroute the flow around a fault. At the same time, as these events would indicate the presence of the fault, the network

[1] http://oss.oetiker.ch/mrtg/

[2] Technically, for any MIB object in an SNMP-managed device.

operator or manager could be alerted about the fault event. Assuming that information from flow context sensors is disseminated in a path-decoupled manner (cf. Section 6.2) over a network with sufficiently redundant paths, then it may still be possible for a management entity to receive feedback from multiple flow context sensors about such an event, which could be aggregated in a way that could help determine the fault's cause and location.

Configuration management deals with provisioning, initialization, and configuration of physical and virtual elements within the network. Extrinsic flow context sensors, including those that can be dynamically deployed to collect device configuration information and even to reconfigure devices (cf. Section 5.2) can be employed for this purpose. Intrinsic flow context can serve as a feedback mechanism to an operator or a management entity, enabling the latter to determine if the current network configuration optimally matches the nature and characteristics of the flows being transported. Additionally, the provisioning of paths, circuits and overlays – also part of configuration management – can be triggered by flow context, either through manual means or as an automatic response to the information contained within flow context tags.

Flow context can be used in accounting management by providing a framework by which numerical flow statistics may be collected and combined with more detailed information regarding the flow, such as the nature of its contents or its anticipated traffic characteristics, to enable a provider to apply an appropriate pricing scheme and to better account for its operational costs. (Such a functionality is provided for instance by `FlowSensor`, described in Section 5.1.) An Internet service provider might want to implement a scheme that applies different prices, costs and markup for ordinary Web browsing, real-time media streaming, peer-to-peer traffic, commercial use, and other traffic classes. Conversely, the information may also be used by a provider to alert users about the potential costs of transmitting these types of traffic, or to offer users various transmission alternatives based on price and performance.

Performance management typically entails gathering network statistics to enable a network manager to determine the network's efficiency, utilization, throughput, error, response times, reliability, capacity, and other performance metrics. While intrinsic flow context conceptually includes the conventional numerical information collected for management purposes (cf. the IPFIX model in Section 4.3.2), the richer amount information available from flow context can be used for more forward-looking and proactive performance management. For example, flow context sensors could be used to sense specific types of traffic, such as peer-to-peer traffic, worm outbreaks, and real-time media flows – all of which may have a different impact on long-term network performance than, for instance, ordinary Web

browsing. The ability to classify flows along these lines and look at their statistics differentially may allow network managers to assess, estimate and plan for capacities appropriate to the growth rates of each of these traffic types, or to plan for and deploy appropriate measures for mitigation [50].

Security management involves the creation and application of security-related policies and the reporting of security-related events. Typically these policies restrict the access to network elements to authorized individuals. Flow context sensors can be used to alert network managers about suspicious flows that might indicate attempts to breach network security, either through unauthorized access, by denial-of-service, or though some other means. A strategic and distributed placement of sensors at vantage points within the network can either facilitate early detection (e.g. by getting an alert from a point upstream or from the network periphery), or ensure that the sensors would not be in the direct line of attack. One interesting complementary approach would be to deploy flow context sensors *away* from the monitored network elements, such as in unused segments of the network's address space, similar to the strategy used in network telescopes, since traffic sent to this space would tend to be spurious as they would be addressed to non-existent hosts on the network [276].

7.7 Chapter summary

This chapter explored some of the potential applications of flow context. It was shown, using simple experiments, how the scheme may be used in implicit QoS signaling, intelligent flow classification and management, and in host mobility and moving networks. Other applications such as in the control and mitigation of malicious or wasteful traffic, and in the various areas of network management, including, fault, configuration, accounting, performance and security management, were examined. Other potential areas to be explored in future work are described in Section 8.3.6.

Chapter 8

Conclusion and Future Work

8.1 Summary and contributions

The convergence of concepts from context-aware computing with adaptive networking has raised the possibility of embedding context-awareness within networks, enabling the realization of context-aware networks. Such networks would be equal partners in human-computer interactions rather than simple packet carriers, offering autonomic properties capable of optimizations not found in current networks [8].

To contribute to the realization of context-aware networks, a number of research objectives were set forth in Section 1.1. These were:

1. Define and develop concepts that may be useful in understanding, designing, implementing, operating and managing context-aware networks;
2. Identify specific entities within the network whose context could be sensed, processed and used;
3. Develop models for the context information describing these network entities;
4. Develop architectural and software components that would allow such context information to be obtained, processed, distributed and used within the network and end-applications; and
5. Investigate illustrative application scenarios for the use of context information within networks.

A major motivation for the work in this thesis is the apparent lack of a strong conceptual foundation in current research efforts to build context-aware networks. The approach in this thesis, and its contribution to the effort, is based on a concept called flow context, defined as any information that can be used to characterize the situation of a flow. Flow context considers flows within the network as entities that represent the interaction between users and networks. A multi-dimensional and integrative approach to its conceptualization was adopted, and as a result, the definition of flow context not only includes properties intrinsic to the flow, but also encompasses matters that are extrinsic to the flow. To further understand and appreciate the concept of flow context, some of its characteristics, such as its dynamism, imperfectness, its multiple representations, and relationships with other types of context, were described.

Since flow context is a novel concept, a formal semantic model was developed, allowing it to be represented, communicated and understood in an unambiguous way to other researchers in the field. This addressed the stated research objective of developing models for information describing entities of contextual interest. An important advantage of the ontological approach adopted in modeling flow context was that the model itself is a machine-processable artifact that could be used during runtime by context-aware networks to handle, interpret, manage or manipulate flow context.

The idea that flow context may exist implicitly, and that it might require some form of sensing, interpretation, processing or sharing within the network led to the definition of the various stages of its notional life cycle.

Both the flow context and flow context lifecycle concepts are seen as contributions towards the objective of developing the conceptual foundations of context-aware networks. The identification of the flow as the main entity whose context should be sensed, processed and used, as well as the development of the idea that its context could be linked to the context of users, networked devices and applications, both address the research objective of identifying specific entities within the network whose context should be sensed and used.

On the other hand, the development and evaluation of various *components* that implement the various stages of the flow context lifecycle addresses the research objective of developing architectural and software components for context-aware networks. These stages were defined as flow context sensing, discovery, aggregation, dissemination, and use, and the remainder of the thesis was devoted to the discussion on each of these stages, as well as some components designed to implement them.

The design and implementation of three different types of sensors that obtain intrinsic and extrinsic flow context was described. The design and implementation of `FlowSensor`, a

modular application that combines flow metering and signature-scanning functionalities, was described. The design and implementation of extrinsic context sensors that allow the location and characteristics of devices and network nodes to be obtained were also described. Aside from the functionality, some design approaches and solutions were contributed, including multi-dimensional sensing, resource reuse, and rapid deployment.

An approach to locating flow context sensors based on distributed hash tables was also evaluated. The results of the evaluation of one class of DHTs called Content-Addressable Networks suggests the possibility of using other DHT protocols, as well as peer-to-peer network architectures based on DHTs, as a scalable, distributed mechanism for locating flow context sensors. Several caching approaches intended to minimize the overhead associated with the use of DHTs were also suggested.

The use of a reasoner for the semantic processing and aggregation of flow context was also investigated. Reasoner-based processing complements the use of ontologies for flow context modeling, as it allows inferences to be made on flow context assertions, based on the domain's semantic model. The performance evaluation of reasoner-based flow context aggregation however yielded some concerns about its computational overhead, and suggested the need to further investigate solutions for optimization and improvement.

To address the final research objective, which was the investigation of application scenarios, this thesis also presented the applications of flow context in mobility, quality of service, flow classification, in controlling or mitigating the effects of malicious or wasteful flows, and in network management. The wide variety of applications demonstrated tends to suggest that the flow context concept may be feasible, useful and potentially powerful mechanism for future context-aware networks.

Despite making some modest contributions and generally addressing all of its stated research objectives, several shortcomings and limitations of the work in this thesis have been identified. In addition, the research also raises new questions and issues that remain open. Section 8.2 discusses some of these issues, while Section 8.3 makes some recommendations for future work, based on both the shortcomings identified during the course of the work, as well as the exciting new possibilities it has uncovered.

8.2 Questions and issues for further consideration

8.2.1 Who benefits from this research?

In Chapter 1 one of the general aims of this work was to enable the creation of context-aware networks, which in turn would contribute to the overall vision of "minimally-distracting networks." The initial vision of "minimal distraction" seems to be a user-centered view, and implies that ultimately the beneficiary of context-aware networking would be the user.

In reality however, network operators and individual users may have different perceptions of what might be best, with the needs of individual users in some cases coming into conflict with those of other users, or with the interests of operators. For example, while users might demand more and more bandwidth for their applications (such as peer-to-peer applications), network operators may tend to regulate their use (or impose a premium) to avoid congesting their links or to accommodate more paying subscribers.

The results from the work in this thesis suggest is that it is possible to construct context-aware networks, that is, networks with architectural support for the sensing, dissemination, aggregation and use of context, in a manner that is agnostic to the intended application. In other words, from a conceptual viewpoint, it is not the provisioning of context-aware components per se that determines who benefits from context-awareness; it is the intended application that determines it. For example, the mobility application demonstrated in Section 7.2 provides some benefit for the user by reducing handoff latency; however, the network operator might not appreciate the significant traffic overhead. On the other hand, the flow classification application presented in Section 7.4 may be beneficial to the network operator who is trying to cope with the onslaught of peer-to-peer traffic for instance, but it may go directly against the interests of the users trying to run them. Achieving a balance between the needs of the various entities within the network, such as individual users, the larger community of users, and the network operator, is a crucial and interesting yet complex question that has not been addressed in this thesis.

Beyond the rather theoretical view that the ultimate beneficiary of context-awareness depends on its particular application, a more practical aspect still remains: are the architectural and operational overheads worth it? Do the additional components, protocols and resources justify the range of optimizations that are enabled? Can this be quantitatively determined or proven? This has not been addressed in this thesis and is left for future work.

8.2.2 Does it violate the end-to-end argument?

In the early 1980s, Saltzer, Reed and Clark of the M.I.T. Laboratory for Computer Science published a paper entitled "End-to-End Arguments in System Design" [179]. They presented what is now a generally-accepted design principle, suggesting that functions or services should not be placed at the lower layers of a system unless these functions or services are needed by all clients of that layer, and that they can be completely implemented in that layer. The end-to-end principle leads to two design goals within the network: (1) application autonomy, where higher-level layers are free to organize lower-level network resources to achieve application-specific goals efficiently, and (2) network transparency, where lower-level layers should only provide resources of broad utility across applications, while providing to applications usable means for effective sharing of resources and resolution of resource conflicts [277].

This leads to the following question: does context-awareness within the network violate the end-to-end principle? In other words, does it either lead to a lesser degree of application autonomy, or less network transparency? A related question might be: should the functionality in networks be kept to the bare minimum of transporting traffic, leaving other functions to the edges?

Similar to the view in the previous section, sensing and distributing context per se do not violate the end-to-end principle. However, certain types of adaptation or other services or actions within the network may be a threat to network transparency, because these may result in unintended or unwanted changes in the flow. Perhaps context tags could selectively specify which adaptation services or functions should apply, or even specify if none should be performed, in order to ensure full network transparency and to preserve end-to-end semantics. For example, the context tag of a media stream could specify if it should not be subjected to lossy compression. On the other hand, certain types of adaptation, such as the rerouting or suppression of excessive, wasteful or malicious traffic (cf. Section 7.5), while causing unintended changes to the flow, may arguably be acceptable to the larger community of users in a network. Although the end-to-end argument certainly had in mind the "effective sharing of resources and resolution of resource conflicts" by end-applications [277], perhaps it did not contemplate the emergence of applications that would not be cooperative in this respect, or worse, those with hostile intentions. As argued in Section 7.5, the failure in some cases by end-systems to effectively deal with these types of flows suggests the need for network-based solutions to address these problems.

It may also be recalled that that one of the arguments presented in early work on network-based adaptation (e.g. [20]) was that it would be more economical in some cases to deploy

adaptation components within the network (which is common to users) than to modify end-applications on (possibly all) users to achieve the same effect. The end-to-end arguments do not prohibit this; they only suggest that the functionality should be implemented completely (i.e. correctly) within that layer or entity. However, network-based adaptation may be implemented correctly only if the information needed to perform the function and to preserve end-to-end semantics (when necessary) can be provided. For example, a network may correctly perform adaptation such as compression on a media flow only if it knows the characteristics of the media payload, and perhaps the media quality levels acceptable to the user – in other words, if the network is fully aware of the context of the flow. Thus, it may be argued that an infrastructure for context-awareness in fact enables network-based adaptation to follow the spirit of the end-to-end arguments, by providing the information necessary for the adaptation to be executed correctly and completely.

8.3 Future work

Because of the novel nature of flow context, much work still has to be done to progress the work described in this thesis. This may involve the development and introduction of new concepts, or a refinement of those that were introduced in this work. Many of the novel concepts and mechanisms introduced here would either still have to be validated further, or transformed into detailed architectural models and implementations for future context-aware networks.

In addition to these, this section discusses some of the gaps identified during the course of this work, and the related issues and approaches that may be investigated in the future.

8.3.1 Sensing

The techniques employed for signature scanning in `FlowSensor` (cf. Section 5.1) were tested only up to a proof-of-concept level. Although the complexity of the BM and BMH algorithms are known [187, 278], it would still be useful to verify their performance on real flows, or at least on packet traces that reasonably replicate the actual packet distributions and content mixes in real flows. An important parameter to consider and evaluate in `FlowSensor`'s performance, aside from its speed, would be its *accuracy* (correctly classifying flows based on detected signatures) as well as its *inaccuracy* (misclassification rate). Such levels of characterization would be useful to system designers who would need to match the resulting performance parameters and resource requirements to their particular application

and deployment scenarios.

In addition, the signature scanning techniques employed may be improved in light of newer and possibly better algorithms. For example, [56] introduces a variant of the BMH algorithm called the Set-wise Boyer Moore Horspool (SBMH) algorithm that can be used in order to search for multiple patterns in parallel, which is something that may enhance the performance of the sensor. Recent work also introduces a new representation for regular expressions called *delayed input deterministic finite automata* (D^2FA) which may be considered for future regex support

Aside from signature scanning, other methods for intrinsic context detection may be integrated into `FlowSensor`. These include various statistical methods, such as the use of statistical (rather than byte pattern) signatures [213], Bayesian analysis techniques [64], and stochastic machine learning techniques [214], for example.

8.3.2 Location

In Section 6.1.1 the use of distributed hash tables in locating flow context sensors was investigated. However, the work in this thesis did not investigate the resource impact (e.g. processing and memory requirements) of using DHTs as well as the practical considerations in deploying such a solution. Although from a performance viewpoint, the use of DHTs seems to be a scalable, decentralized solution, the method does present some overhead in terms of actual network-layer routing delay, as one hop in the DHT overlay may translate to multiple hops within the physical network. The actual delay at the level of the network (rather than the overlay) was not characterized in this thesis.

To mitigate some of the effects of routing overhead, the use of various caching mechanisms and strategies may be investigated in the future. Since the ConCoord-DHT model stores either sensor addresses or context client addresses rather than the context information itself, the relatively static nature of the $<key, value>$ pairs would lead well to caching, even in the face of numerous changes within the overlay topology. Caching recently-obtained $<key, value>$ pairs by overlay nodes has been suggested in [236], resulting in what might be analogous to "non-authoritative replies," and local caching may also be done by context clients themselves. A time-to-live (TTL) parameter could be specified by context sources during registration to provide a metric for the expiration of cache entries. A `get` may also be simultaneously routed within the overlay, even in the event of a cache hit, for the purpose of validating or invalidating a cached entry. Various cache coherency protocols may also be investigated.

Another approach worth investigating would involve the use of cached source addresses of overlay nodes obtained from transaction replies or acknowledgments sent from the overlay to a client. For example, a successful `RESOLVE` request should be followed by a `RESOLVEACK` reply from the overlay node to the requesting client. If the client needs to contact the overlay again because its cached data has already expired, it could now use the cached address of the overlay node as the initial known node. The hypothesis is that even in the event that the cached address of an overlay node no longer validly maps to a key (e.g. perhaps because that portion of its zone has already been assigned to another node), that node may, with high probability, be located in within the immediate vicinity of the new owner of the target zone. Thus there would be a performance advantage if an external node would issue the `put` or a `get` request using the cached node's address as the known node.

Further functional enhancements to CAN, as suggested in [236] such as replication, multiple realities, zone overloading, and others, may be investigated in future versions of the simulator.

8.3.3 Flow context modeling

The ontology-based modeling work for flow context in this thesis should have ideally been done in a collaborative manner with domain experts, so that the model may gain acceptance within the wider research community. Since the ontology developed in this thesis was solely developed by the author (although effort was made to reuse other relevant ontologies that have been or are being standardized), the next step should be for other researchers in the domain to examine, critique, modify, or contribute to it. In addition, there is a need to validate the model – in other words, to answer the question, "How well does the model reflect reality?" The solution to the issue of validation also partly lies within the peer-review and collaborative process: the more the ontology becomes a *shared conceptualization* of the domain, the better the chance that it accurately reflects the real world.

Practical and system-related issues related to the management and use of ontologies in future context-aware networks, while discussed in Section 7.1.4, were not thoroughly nor experimentally investigated in this thesis. For example, while the flow context ontology was fully used in an online mode to test the performance of flow context aggregation in Section 6.3.2, it was not used in an online mode for the demonstrations in Chapter 7. As a result, the latter demonstrations did not reflect the potential performance impacts of using the ontology in an online mode in those application scenarios. This would have helped identify those applications where the online use of the ontology-based model would

be practical. Aside from performance issues, the deployment of knowledge-based solutions, such as in future context-aware networks, should take into consideration important practical issues regarding the size and complexity of ontologies and their storage; transmission overhead; mechanisms for updating and managing them; components for discovering them and making them accessible (ontology services); dealing with different domain-specific ontologies; and others. These issues were not experimentally investigated in this thesis.

Another shortcoming of the work in this thesis concerns the linkage of user-related context with the ontology developed for flow context. It may be recalled that a high-level aim of the work was to enable the creation of "minimally-distracting networks" which somewhat implies network optimization that takes into account the situation (context) or even the experience of the user. Although Chapter 4 argues that the *focus* of the work in this thesis would be on developing the flow-related terms in the ontology, a more explicit linkage to and reuse of ontologies describing user-related context would have been useful, and is therefore strongly recommended for future work.

8.3.4 Aggregation

In Section 6.3.2, the use of a reasoner for flow context processing and aggregation was evaluated, and its performance was found to be inadequate for real-time flow context aggregation. As discussed in Section 6.3.2, this severely limits the applicability of reasoner-based aggregation to certain scenarios, such as for small networks, or where the context has been pre-aggregated, or where the flows are relatively long-lived. Despite these less-than-encouraging results, or perhaps in view of these, there is a need to continuously improve the techniques involved in this approach as well as the methods used to evaluate it. The following may be considered to improve the evaluation methodology for future work:

1. The queries tested in Section 6.3.2 were very simple, and did not demonstrate other more "creative" kinds of inferences that are possible with the reasoner. While it was partly due to circumstance (limited information from traces), it was also a matter of design: it would be easy for others to compare the performance of other algorithms that performed the same type of (simple) aggregation. Nonetheless, it would be interesting to experiment with other classes of context assertions in the future

2. Since the evaluation methodology only used some of the platform-specific mechanisms and optimizations, further experimentation with other modes of query optimization are suggested. However, even with a set of comprehensive tests on one platform, it would be incorrect to form conclusions about the performance of reasoners *in*

general. Thus, there is a need to perform similar tests on other reasoner architectures and platforms. In general, there is also an obvious need to experiment with other hardware and operating system configurations as well.

3. The tests only portrayed a monotonically increasing number of flows. In reality, as new flow assertions are added, other flow assertions need to be removed (using nRQL `forget` statements) to model flows disappearing or timing out. It would also useful to observe "quasi-steady states" where there may be a short-term equilibrium in the number of new and expiring flows, where, even with a constant number of flow instances (and approximately uniform memory utilization), there are still constant changes within in the ABox because of the replacement of old assertions by new ones.

4. Timestamp information in packet traces can be used to create more realistic simulations of packet arrival rates, which in turn would better simulate actual rates of flow instance creation and flow context assertions. Different traces from networks of different sizes may be used to simulate a wider range of conditions.

5. A more comprehensive evaluation of reasoning-based flow context aggregation should at least use full traces (non-anonymized headers + full packet payloads). This would also provide an environment to test the performance of `FlowSensor` as well. The best form of testing and evaluation, of course, would be on a live network, with real flows.

In addition to improving the evaluation of reasoner-based flow context aggregation, other modes of aggregation could be explored in future work. These could include:

- *The use of persistent storage for flow context.* The mechanisms and protocols for the distributed and scalable storage of flow context could be investigated. For instance, distributed hash tables could be considered for this purpose. It should be recalled however that the storage of flow context is a critical issue in system design (discussed briefly in Section 7.1.3) that deserves detailed examination in future work.

- *Topology-based aggregation.* Context could be aggregated based on the network topology. For example, Lim and Stadler [279] transformed the topology of the network into an execution graph and propagated management queries on network nodes along this graph. The results of these queries were then back-propagated, in a process called contraction, and aggregated along the same graph. This could also be explored for the aggregation of flow context information from sensors.

- *Process flow-based aggregation* The aggregation of flow context could be specified on the basis of logical workflows or process flows. For example, Reichert, Kleis and Giaffreda [233] proposed the use of *multi-pipes*, which were directed acyclic graphs specifying a sequence of aggregation and processing operations for context information. This mode of specifying composition and aggregation can be explored for application to flow context as well.

8.3.5 The need for validation

Aside from the need to validate the ontology model (as mentioned in Section 8.3.3), there is a need to further validate the architectural and software components presented in this thesis, especially in practical and real deployment scenarios. The approach taken in this thesis was to develop and examine the components in detail, often in isolation from each other. To demonstrate the application of the flow context *concepts*, a limited subset of the functionalities were used in an integrated manner. There is a need therefore to further test the full components in an integrated fashion (formally defining the inter-component interfaces and protocols in the process) and subject them to experimental scenarios that reflect real-life application loads, usage, issues and problems. For example, the flow classification experimental scenario discussed in Section 7.4 could be validated in a more realistic way by (1) integrating a full version of `FlowSensor` developed in Section 5.1, (2) using a relevant subset of the ontology in an online manner, (3) performing flow context aggregation simultaneously (taking into account the improvements suggested in Section 8.3.4, and (3) deploying it on a live network with real user traffic. Additionally, sensors for user-level context could be deployed to try to provide further contextual input for the aggregation process.

Although the work in this thesis was aimed at making a contribution to the study of context-aware networks, one of the assumptions made was that context-aware networks would help create "minimally-distracting" or "invisible" networks. The discussion in Section 8.2.1 however points out that context-awareness within the network may either benefit users, or network operators, or other entities, based on the way the context is used. Therefore, there may be cases where context-awareness may result in network behavior that is not "invisible" nor "minimally-distracting." Nonetheless, it would still be useful to validate in a rigorous way whether context-aware networks can indeed help build invisible networks. Such a validation would perhaps require user context to be more explicitly modeled (as stated in Section 8.3.3), and an experimental method be developed to gauge the quality of the user's (subjective) experience, or to quantify the usefulness of the optimization or adaptation received by the user's flow.

8.3.6 Applications

While the application areas described in Chapter 7 could be further developed and validated in future work, other applications areas should also be explored.

For example, flow context may potentially be used for overlay routing and content delivery. Requests for content streams and the corresponding delivered content may be classified and routed through a network based on flow context tags. In the case of multimedia or real-time streams, the flow that contains the content request may contain a description of QoS requirements that the underlying network may use as a basis for a routing decision or to map the flow to an appropriate overlay. In the reverse direction, the flow containing the content to be delivered may contain a description of both the requirements of the requestor and the characteristics of the content, again for routing or overlay mapping purposes.

Flow context might also be used in overlays, particularly in structured peer-to-peer networks, to match flow requirements (e.g. the geographic location of a flow's destination) with sensed knowledge about the relative proximity of peer nodes or information about the topology of the underlying physical network, such as in topology-aware routing [280] and topology-aware server selection [281].

8.3.7 Security, privacy and trust

Section 7.6 briefly mentioned some applications of context tags in enhancing network security. While flow context might be useful in this respect, it can potentially be a double-edged sword. There is also the possibility that flow context may be misused, deliberately or otherwise, to compromise network security, disrupt the network's proper operation, or for other malicious purposes. The emergence of new modes of context sensing and processing may also spawn new modes of attack: perhaps instead of simple buffer exploits and overload attacks, these new modes would consist more of algorithmic attacks [56] or malicious training [282] deliberately targeted at the sensing infrastructure. In the future, "semantic exploits" that take advantage of ontological inaccuracies or flawed reasoner architectures, and that intend to induce unintended inferences or to degrade reasoner performance, might even appear on the horizon.

There may be a need to protect communication between context sensors and context-sensitive nodes and to develop mechanisms for this purpose. This might include mechanisms that would prevent context tags from being forged, or for existing tags in a flow to be altered in a malicious way. Even in absence of malicious intent, the issue of *safety* is

a concern: can corrupted tags, or even well-formed ones, produce unwanted or undesired effects within the network? How can this be prevented or mitigated?

Privacy is still a wide-open issue in pervasive and ubiquitous computing in general, and any system that senses and uses context is faced with the same challenge of safeguarding user privacy. As this feature was identified as one of the key features of minimally-distracting and invisible networks, it would be desirable to explore some mechanisms to assure this. Such mechanisms might include, among others, the use of encryption, an authorization framework [283], or the use of *obfuscation* [284] to control the level of information made available within a context-aware network. Another mechanism might be the introduction of *incentives* that might encourage users to sacrifice some level of privacy in exchange for preferential treatment or service levels within the network.

8.3.8 Quality of context

Section 3.3.3 mentioned that one of the characteristics of flow context is its (potential) imperfectness, which some authors attempt to express in terms of quality of context (QoC). Although quality of context may be used to model some objective and subjective quality attributes of context information, such as its fidelity, precision, accuracy, trustworthiness, resolution, frequency, and timeliness, it had not been modeled in the flow context ontology nor used in the applications of flow context in this thesis. This is an important issue that has to be addressed in future work.

Quality of context was also briefly mentioned in Section 2.3.2 within the framework of context-level agreements (CLAs), where entities pre-negotiate QoC levels *before* exchanging context information. An alternative would be to consider QoC as a mechanism to signal the *utility* perceived or assigned by a user or receiver of flow context information, *after* it had started to receive the information. This would be important in determining the amount and quality of context a sensor should provide, and could be viewed as a feedback loop from context user to context provider. In such a context signaling protocol, for instance, the absence of positive feedback from context users could be construed as a signal to a context source to decrease or even terminate the transmission of flow context information, especially in context push or event notification scenarios.

References

[1] R. Katz and E. Brewer. The Case for Wireless Overlay Networks. *Proceedings of the SPIE Multimedia and Networking Conference (MMNC'96)*, San Jose, CA, USA, January 1996, pp. 77–88.

[2] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3), September 1991, pp. 94–104.

[3] M. Weiser. The World is not a Desktop. *Interactions*, January 1994, pp. 7–8.

[4] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4), August 2001, pp. 10–17.

[5] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2), April 2002, pp. 22–31.

[6] A. K. Dey, D. Salber, and G. D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction (HCI) Journal*, 16 (2-4), 2001, pp. 97–166.

[7] K. Yang. On an All-Policy Way to Active Context-Aware Services. Unpublished PhD transfer report, Department of Electronic and Electrical Engineering, University College London, August 2003.

[8] S. Dobson. Putting Meaning into the Network: Some Semantic Issues for the Design of Autonomic Communication Systems. *1st IFIP Workshop on Autonomic Communications*, Berlin, Germany, October 2004, pp. 207–216.

[9] A. Karmouch, A. Galis, R. Giaffreda, T. Kanter, A. Jonsson, A. Karlsson, R. Glitho, M. Smirnov, M. Kleis, C. Reichert, A. Tan, M. Khedr, N. Samaan, H. Laamanen, M. El Barachi and J. Dang. Contextware Research Challenges in Ambient Networks. *1st*

International Workshop on Mobility Aware Technologies and Applications, October 2004, pp. 62–77.

[10] A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis, College of Computing, Georgia Institute of Technology, December 2000. Available at http://www.cc.gatech.edu/fce/ctk/pubs/dey-thesis.pdf; accessed December 2006.

[11] A. Schmidt. *Ubiquitous Computing - Computing in Context.* PhD thesis, Computing Department, Lancaster University, November 2002. Available at http://www.comp.lancs.ac.uk/~albrecht/phd/Albrecht_Schmidt_PhD-Thesis_Ubiquitous-Computing_print1.pdf; accessed December 2006.

[12] G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. *Darthmouth Computer Science Technical Report TR2000-381*, Darthmouth College, November 2000. Available at ftp://ftp.cs.dartmouth.edu/TR/TR2000-381.ps.Z; accessed December 2006.

[13] S. Meyer and A. Rakotonirainy. A Survey of Research on Context-Aware Homes. *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003*, Adelaide, Australia, Vol. 21, February 2003, pp. 159–168.

[14] J. Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. *Proc. 2nd International Symposium on Wearable Computers*, Pittsburgh, PA, USA, October 1998, pp. 92–99.

[15] M. Wallbaum. WhereMoPS: An Indoor Geolocation System. *The 13th IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications*, Lisbon, Portugal, Vol. 4, September 2002, pp. 1957–1961.

[16] J. Hightower, B. Brumitt, and G. Borriello. The Location Stack: A Layered Model for Location in Ubiquitous Computing. *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, Callicoon, NY, June 2002, pp. 22–28.

[17] R. Ocampo and H. De Meer. Smart Wireless Access Points for Pervasive Computing. *Proc. First Working Conference on Wireless On Demand Network Systems (WONS '04)*, Lecture Notes in Computer Science LNCS 2928, January 2004, pp. 329–343.

[18] A. Campbell, G. Coulson and D. Hutchison. A Multimedia Enhanced Service in a Quality of Service Architecture. *Proc. 4th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'93)*, Lancaster, U.K., October 1993, pp. 124-137

[19] N. Yeadon, F. Garcia, A. Campbell and D. Hutchison. QoS Adaptation and Flow Filtering in ATM Networks. *Proc. 2nd International Workshop on Advanced Teleservices and High-Speed Communication Architectures*, Heidelberg, Germany, September 1994, pp. 191–203.

[20] A. Fox, S. D. Gribble, Y. Chawathe and E. A. Brewer. Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives. *IEEE Personal Communications Special Issue on Adapting to Network and Client Variability*, 5(4), August 1998, pp. 10–19.

[21] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions Part Two: Media Types. *Request for Comments 2046*, November 1996. Available at http://www.ietf.org/rfc/rfc2046.txt; accessed December 2006.

[22] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn and K. Walker. Agile Application-Aware Adaptation for Mobility. *Proc. 16th ACM Symposium on Operating System Principles*, Saint Malo, France, November 1997, pp. 276–287.

[23] M. Yarvis, P. Reiher and G. Popek. Conductor: A Framework for Distributed Adaptation. *Proc. 7th Workshop on Hot Topics in Operating Systems*, Arizona, USA, March 1999, pp. 44–49.

[24] P. Sudame and B. R. Badrinath. Transformer Tunnels: A Framework for Providing Route-Specific Adaptations. *USENIX Annual Technical Conference (NO 98)*, New Orleans, Louisiana, USA, June 1998, pp. 191–200.

[25] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System Support for Bandwidth Management and Content Adaptation in Internet Applications. *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, San Diego, California, October 2000, pp. 213–226.

[26] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. *Request for Comments 2616*, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt; accessed December 2006.

[27] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks and ISDN*, 17(1), June 1989, pp. 1–14.

[28] M. Anagnostou, M. Lambrou and E. Sykas. Context Aware Service Engineering in Support of Future Business Networks. *2nd Context Aware Collaborative Environments for Next Generation Business Networks (COCONET)*

Workshop, Helsinki, Finland, December 2002. Presentation slides available at http://context.upc.es/Papers/ContextCOCONET.pdf; accessed December 2006.

[29] K. Jean, K. Yang and A. Galis. A Policy Based Context-Aware Service for Next Generation Networks. *Proceedings of the 8th London Communications Symposium*, London, U.K., September 2003, pp. 349–352.

[30] N. Niebert, A. Schieder, H. Abramowicz, G. Malmgren, J. Sachs, U. Horn, C. Prehofer, and H. Karl. Ambient Networks: An Architecture for Communication Networks Beyond 3G. *IEEE Wireless Communications*, 11(2), April 2004, pp. 14–22.

[31] A. Jonsson, R. Giaffreda, M. Barachi, R. Glitho, F. Belqasmi, M. Smirnov, M. Kleis, C. Reichert, A. Karmouch, M. Khedr, A. Karlsson, H. Laamanen, H. Helin, A. Galis, R. Ocampo and J. Zhang. Ambient Networks ContextWare: First Paper on Context-Aware Networks. *Deliverable Report D-6-1, Ambient Networks Project*, January 2005. Document number IST-2002-507134-AN/WP6/D61, available at http://www.ambient-networks.org/phase1web/publications/D6-1_PU.pdf, accessed December 2006.

[32] K. Henricksen, J. Indulska and A. Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. *Proceedings of the First International Conference on Pervasive Computing*, Zurich, Switzerland, August 2002, pp. 167–180.

[33] F. Sestini, M. Smirnov, I. Stavrakakis, S. Dobson, A. Kalis, W. van de Velde, J. Fonollosa and K. David. Situated and Autonomic Communications. *Summary Report*, May 2004. Available at ftp://ftp.cordis.lu/pub/ist/docs/fet/comms-60.pdf; accessed December 2006.

[34] B. Schilit and M. Theimer. Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5), September 1994, pp. 22–32.

[35] P. Brown, J. Bovey and X. Chen. Context-Aware Applications: From the Laboratory to the Marketplace. *IEEE Personal Communications*,4(5), October 1997, pp. 58–64.

[36] A. Schmidt, M. Beigl, and H. Gellersen. There is More to Context Than Location. *Computers and Graphics Journal*, 23(6), December 1999, pp. 893–901.

[37] A. Schmidt, K. van Laerhoven. How to Build Smart Appliances? *IEEE Personal Communications*, 8(4), August 2001, pp. 66–71.

[38] R. Giaffreda, A. Karmouch, A. Jonsson, A. Karlsson, M. Smirnov, R. Glitho, and A. Galis. Context-Aware Communication in Ambient Networks. *Wireless World Research*

Forum 11th Meeting, Oslo, Norway, June 2004. Available at http://www.ambient-networks.org/docs/Context_aware_Communication_in_Ambient_Networks.pdf; accessed December 2006.

[39] J. Case, R. Mundy, D. Partain and B. Stewart. Introduction and Applicability Statements for Internet Standard Management Framework. *Request for Comments 3410*, December 2002. Available at http://www.ietf.org/rfc/rfc3410.txt; accessed December 2006.

[40] D. Clark. The Design Philosophy of the DARPA Internet Protocols. *ACM SIGCOMM Computer Communication Review*, 18(4), August 1988, pp. 106–114.

[41] R. Braden, D. Clark, and S. Shenker. Integrated Services Architecture in the Internet: an Overview. *Request for Comments 1633*, June 1994. Available at http://www.ietf.org/rfc/rfc1633.txt, accessed December 2006.

[42] K. Claffy, H. Braun and G. Polyzos. A Parameterizable Methodology for Internet Traffic Flow Profiling. *IEEE Journal on Selected Areas in Communications*, 13(8), October 1995, pp. 1481–1494.

[43] N. Brownlee and M. Murray. Streams, Flows and Torrents. *Proc. 2nd Passive and Active Measurement Conference (PAM2001)*, Amsterdam, April 2001. Available at www.ripe.net/pam2001/Papers/talk_07.ps.gz; accessed December 2006.

[44] N. Brownlee. Understanding Internet Traffic Streams: Dragonflies and Tortoises. *IEEE Communications Magazine*, 40(10), October 2002, pp. 110–117.

[45] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification. *Request for Comments 2205*, September 1997. Available at http://www.ietf.org/rfc/rfc2205.txt; accessed December 2006.

[46] Work package 1, EU-IST Ambient Networks Project. AN Framework Architecture. *Deliverable Report D-1-5*, Ambient Networks Project, December 2005. Document number IST-2002-507134-AN/WP1-D05, available at http://www.ambient-networks.org/phase1web/publications/D1_5_AN_Framework_Architecture_PU.pdf; accessed December 2006.

[47] J. Quittek, T. Zsesby, B. Claise and S. Zander. Requirements for IP Flow Information Export (IPFIX). *Request for Comments 3917*, October 2004. Available at http://www.ietf.org/rfc/rfc3917.txt; accessed December 2006.

[48] N. Brownlee. Using NetTraMet for Production Traffic Measurement. *Proc. 2001 IEEE/IFIP International Symposium on Integrated Network Management*, Seattle, WA, USA, May 2001, pp. 213–226.

[49] N. Brownlee, C. Mills and G. Ruth. Traffic Flow Measurement: Architecture. *Request for Comments 2722*, October 1999. Available at http://www.ietf.org/rfc/rfc2722.txt; accessed December 2006.

[50] T. Karagiannis, P. Rodriguez and K. Papagiannaki. Should Internet Service Providers Fear Peer-Assisted Content Distribution? *ACM Internet Measurement Conference (IMC 2005)*, Berkeley, CA, USA, October, 2005, pp. 63–76.

[51] R. Katz, G. Porter, S. Shenker, I. Stoica and M. Tsai. COPS: Quality of Service vs. Any Service at All. *Proceedings of the 13th International Workship on Quality of Service (IWQoS 2005)*, Passau, Germany, June 2005, pp. 3–15.

[52] S. Sen, O. Spatscheck and D. Wang. Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures. *13th International Conference on World Wide Web (WWW 2004)*, NY, USA, 2004, pp. 512–521.

[53] K. Gummadi, R. Dunn, S. Sariou, S. Dribble, H. Levy and J. Zahorjan. Measurement, Modeling and Analysis of a Peer-to-Peer File-Sharing Workload. *19th ACM Symposium on Operating Systems Principles (SOSP-19)*, October 2003, pp. 314–329.

[54] G. Cormack and T. Lynam. On-line Supervised Spam Filter Evaluation. Available at http://plg.uwaterloo.ca/~gvcormac/spamcormack06.pdf; accessed December 2006.

[55] M. Roesch. Snort – Lightweight Intrusion Detection for Networks. *USENIX 13th Systems Administration Conference*, Seattle, USA, November 1999, pp. 229–238.

[56] M. Fisk and G. Varghese. Fast Content-Based Packet Handling for Intrusion Detection. *UCSD Technical Report CS2001-0670*, University of California at San Diego, May 2001. Available at http://woozle.org/~mfisk/papers/ucsd-tr-cs2001-0670.pdf; accessed December 2006.

[57] A. Kumar, V. Paxson and N. Weaver. Exploiting Underlying Structure for Detailed Reconstruction of an Internet-Scale Event. *ACM Internet Measurement Conference 2005 (IMC 2005)*, Berkeley, CA, USA, October 2005. Available at http://www.imconf.net/imc-2005/papers/imc05efiles/kumar/kumar.pdf; accessed December 2006.

[58] W. Leland, M. Taqqu, W. Willinger and D. Wilson. On the Self-Similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Transactions on Networking*, 2(1), February 1994, pp. 1–15.

[59] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3), June 1995, pp. 226-224.

[60] M. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5(6), December 1997, pp. 835–846.

[61] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. *Proceedings of the 2001 ACM SIGCOMM Internet Measurement Workshop*, San Francisco, CA, USA, November 2001, pp. 75–80.

[62] J. Jedwab, P. Phaal, B. Pinna. Traffic Estimation for the Largest Sources on a Network, Using Packet Sampling with Limited Storage. *HP Labs Technical Reports HPL-92-35*, HP Laboratories Bristol, March 1992. Available at http://www.hpl.hp.com/techreports/92/HPL-92-35.html; accessed December 2006.

[63] K. Claffy, G. Polyzos, and H-W. Braun. Application of Sampling Methodologies to Network Traffic Characterization. *ACM SIGCOMM '93*, San Francisco, CA, USA, September 1993, pp. 194–203.

[64] A. Moore and D. Zuev. Internet Traffic Classification Using Bayesian Analysis. *Proceedings of ACM SIGMETRICS'05*, Banff, Canada, June 2005, pp. 50–60.

[65] A. Schmidt, K. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven and W. Van de Velde. Advanced Interaction in Context. *1st International Symposium on Handheld and Ubiquitous Computing (HUC99)*, Karlsruhe, Germany, September 1999, pp. 89–101.

[66] P. Gray and D. Salber. Modelling and Using Sensed Context Information in the Design of Interactive Applications. *8th IFIP International Conference on Engineering for Human-Computer Interaction (EHCI 2001)*, Toronto, Canada, May 2001, pp. 317–336.

[67] M. Ebling, G. Hunt and H. Lei. Issues for Context Services for Pervasive Computing. *Workshop on Middleware for Mobile Computing*, Heidelberg, Germany, November 2001. Available at http://www.csse.monash.edu.au/~phaghigh/Ebling_Hunt_Lie.pdf; accessed December 2006.

[68] P. Castro, P. Chiu, T. Kremenek, and R. Muntz. A Probabilistic Room Location Service for Wireless Networked Environments. *3rd International Conference on Ubiquitous Computing (UbiComp 2001)*, Atlanta, Georgia, USA, September 2001, pp. 18–34.

[69] T. Buchholz, A. Küpper and M. Schiffers. Quality of Context: What It Is and Why We Need It. *Workshop of the HP OpenView University Association 2003 (HPOVUA 2003)*, Geneva, Switzerland, July 2003. Available at http://www.mnm-team.org/projects/kodi/publications/buks03.ps; accessed December 2006.

[70] M. Huebscher and J. McCann. Adaptive Middleware for Context-Aware Applications in Smart-Homes. *2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2004)*, Toronto, Ontario, Canada, October 2004, pp. 111–116.

[71] M. Razzaque, S. Dobson and P. Nixon. Categorization and Modelling of Quality in Context Information. *Proc. IJCAI 2005 Workshop on AI and Autonomic Communications*, Edinburgh, Scotland, August 2005. Available at http://www.csi.ucd.ie/UserFiles/publications/1124274826156.pdf; accessed December 2006.

[72] B. Siljee, I. Bosloper and J. Nijhuis. A Classification Framework for Storage and Retrieval of Context. *1st International Workshop on Modeling and Retrieval of Context (MRC 2004)*, Germany, September 2004. Available at http://CEUR-WS.org/Vol-114/; accessed December 2006.

[73] P. Brown. The Stick-e Document: A Framework for Creating Context-Aware Applications. *Proceedings of Electronic Publishing 1996 (EP'96)*, September 1996, pp. 259–272.

[74] R. Ocampo, A. Galis and C. Todd. Triggering Network Services Through Context-Tagged Flows. *Proceedings of the 2nd International Workshop on Active and Programmable Grids Architectures and Components (APGAC'05) / International Conference on Computational Science 2005 (ICCS 2005).* Lecture Notes in Computer Science (LNCS) 3516, Atlanta, USA, May 2005, pp. 259–266.

[75] R. Ocampo, A. Galis, H. De Meer and C. Todd. Implicit Flow QoS Signaling Using Semantic-Rich Context Tags. *Proceedings of the 13th International Workship on Quality of Service (IWQoS 2005).* Lecture Notes in Computer Science (LNCS) 3552, Passau, Germany, June 2005, pp. 369–371.

[76] R. Ocampo, A. Galis, H. De Meer and C. Todd. Supporting Mobility Adaptation Through Flow Context. Short paper proceedings of the *2nd International Workshop on Mobility Aware Technologies and Applications (MATA 2005)*, Montreal, Canada, October 2005, pp. 16–20.

[77] R. Ocampo, A. Galis, H. De Meer and C. Todd. Flow Context Tags: Concepts and Applications. *IFIP TC6 Conference on Network Control and Engineering for QoS, Security and Mobility (NetCon'05)*, Lannion, France, November 2005.

[78] D. Clark, C. Partridge, J. C. Ramming and T. Wroclawski. A Knowledge Plane for the Internet. *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, Karlsruhe, Germany, August 2003, pp. 3–10.

[79] T. Dietterich and P. Langley. Machine Learning for Cognitive Networks: Research Challenges (Draft of May 11, 2003). Available at http://web.engr.oregonstate.edu/~tgd/kp/dl-report.pdf; accessed December 2006.

[80] J. Hong and J. Landay. A Context/Communication Information Agent. *Personal Technologies (Special Issue on Situated Interaction and Context-Aware Computing)*, 5(1), 2001, pp. 78–81.

[81] A. Schmidt, A. Takaluoma and J. Mäntyjärvi. Context-Aware Telephony Over WAP. *Personal Technologies*, 4(4), December 2000, pp. 225–229.

[82] C. Schmandt, N. Marmasse, S. Marti, N. Sawhney and S. Wheeler. Everywhere Messaging. *IBM Systems Journal*, 39(3-4), July 2000, pp. 660–677.

[83] A. Ranganathan, R. Campbell, A. Ravi and A. Mahajan. ConChat: A Context-Aware Chat Program. *IEEE Pervasive Computing*, July-Sept 2002, pp. 52–58.

[84] A. Ranganathan and H. Lei. Context-Aware Communication. *IEEE Computer*, 36(4), April 2003, pp. 90–92.

[85] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, 1(4), October 2002, pp. 74–83.

[86] S. Yau, F. Karim, Y. Wang, B. Wang and S. Gupta. Reconfigurable Context-Sensitive Middleware for Pervasive Computing. *IEEE Pervasive Computing*, 1(3), July 2002, pp. 33–40.

[87] S. Yau and F. Karim. An Adaptive Middleware for Context-Sensitive Communication for Real-Time Applications in Ubiquitous Computing Environments. *Real-Time Systems*, 26(1), January 2004, pp. 29–61.

[88] S. Yau and F. Karim. A Lightweight Middleware Protocol for Ad Hoc Distributed Object Computing in Ubiquitous Computing Environments. *Proceedings of the Sixth*

IEEE International Symposium on Real-Time Distributed Computing (ISORC'03), Hokkaido, Japan, May 2003, pp. 172–179.

[89] M. Khedr and A. Karmouch. Exploiting Agents and SIP for Smart Context Level Agreements. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Canada, Vol. 2, August 2003, pp. 522–525.

[90] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. SIP: Session Initiation Protocol. *Request for Comments 3261*, June 2002. Available at http://www.ietf.org/rfc/rfc3261.txt; accessed December 2006.

[91] X. Wang, D. Zhang, T. Gu and H. Pung. Ontology Based Context Modeling and Reasoning Using OWL. *2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW'04)*, Orlando, Florida, USA, March 2004, pp. 18–22.

[92] T. Strang and C. Linnhoff-Popien. A Context Modeling Survey. *6th International Conference on Ubiquitous Computing*, Nottingham, U.K., September 2004. Available at http://www.mobile.ifi.lmu.de/common/Literatur/MNMPub/Publikationen/stli04a/PDF-Version/stli04a.pdf; accessed December 2006.

[93] D. Balakrishnan, M. El Barachi, A. Karmouch and R. Glitho. Challenges in Modeling and Disseminating Context Information in Ambient Networks. *2nd International Workshop on Mobility Aware Technologies and Applications (MATA 2005)*, Montreal, Canada, October 2005, pp. 32–42.

[94] M. Handley, V. Jacobson. SDP: Session Description Protocol. *Request for Comments 2327*, April 1998. Available at http://www.ietf.org/rfc/rfc2327.txt; accessed December 2006.

[95] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. *Request for Comments 1889*, January 1996. Available at http://www.ietf.org/rfc/rfc1889.txt; accessed December 2006.

[96] H. Schulzrinne. RTP Profile for Audio and Video Conferences with Minimal Control. *Request for Comments 1890*, January 1996. Available at http://www.ietf.org/rfc/rfc1890.txt; accessed December 2006.

[97] S. Yau and F. Karim. Context-Sensitive Middleware for Real-time Software in Ubiquitous Computing Environments. *4th IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC 2001)*, Magdeburg, Germany, May 2001, pp. 163–170.

[98] J. Indulska, R. Robinson, A. Rakotonirainy and K . Henricksen. Experiences in Using CC/PP in Context-Aware Systems. *4th International Conference on Mobile Data Management (MDM2003)*, Melbourne, Australia, January 2003, pp. 247–261.

[99] B. Schilit, M. Theimer and B. Welch. Customizing Mobile Applications. *USENIX Mobile and Location-Independent Computing Symposium*, Massachusetts, USA, August 1993, pp. 129–138.

[100] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau (editors). Extensible Markup Language (XML) 1.1 *W3C Recommendation 04 February 2004, edited in place 15 April 2004*. Available at http://www.w3.org/TR/xml11/; accessed December 2006.

[101] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML-Based Quality of Service Enabling Language for the Web. *Journal of Visual Language and Computing (JVLC), Special Issue on Multimedia Languages for the Web*, 13(1), February 2002, pp. 61–95.

[102] International Organization for Standardization JTC 1. Information Technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2. *International Standard ISO/IEC 19501*, Reference number ISO/IEC 19501:2005(E), April 2005.

[103] P. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1), March 1976, pp. 9–36.

[104] D. Beckett (editor). RDF/XML Syntax Specification (Revised). *W3C Recommendation 10 February 2004*. Available at http://www.w3.org/TR/rdf-syntax-grammar/; accessed December 2006.

[105] J. Serrano O., J. Serrat F., K. Yang and E. Salamanca C. Modelling Context Information for Managing Pervasive Network Services. *International Conference on Modelling and Simulation (ICMS'05)*, Marrakech, Morocco, November 2005. Available at http://privatewww.essex.ac.uk/~kunyang/publications/2005/ICMS05-contextModelling.pdf; accessed December 2006.

[106] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider and L. Stein. OWL Web Ontology Language Reference. *W3C Recommendation 10 February 2004*. Available at http://www.w3.org/TR/owl-ref/; accessed December 2006.

[107] A. Shehzad, H. Ngo, K. Pham and S. Lee. Formal Modeling in Context Aware Systems. *1st International Workshop on Modeling and Retrieval of Context (MRC 2004),*

Germany, September 2004. Available at http://CEUR-WS.org/Vol-114/; accessed December 2006.

[108] J. Serrano, J. Serrat and A. Galis. Ontology-Based Context Information Modelling for Managing Pervasive Applications. *International Conference on Autonomous and Autonomic Systems (ICAS'06)*, San Jose, CA, USA, July 2006, pp. 47–53.

[109] R. Giaffreda, J. Dang, R. Glitho, M. E. Barachi, F. Belqasmi, J. Mattam, T. Kanter, C. Reichert, M. Smirnov, A. Karmouch, D. Balakrishnan, H. Harroud, A. Karlsson, H. Laamanen, M. Laukkanen, R. Ocampo, K. Jean and A Galis. Ambient Networks ContextWare: Second Paper on Context-Aware Networks. *Deliverable Report D-6-3, Ambient Networks Project*, December 2005. Document number IST-2002-507134-AN/WP6/D63. Available at http://www.ambient-networks.org/phase1web/publications/D6_3_Ambient_Networks_ContextWare_Second_Paper_on_Context-Aware_Networks_PU.pdf, accessed December 2006.

[110] M. Smith, C. Welty and D. McGuinness. OWL Web Ontology Language Guide. *W3C Recommendation 10 February 2004*. Available at http://www.w3.org/TR/owl-guide/; accessed December 2006.

[111] D. McGuinness. Ontologies Come of Age. *Spinning the Semantic Web: Bringing the World Wide Web to its Full Potential*, MIT Press, 2003, pp. 171–194.

[112] L. Obrst. Ontologies for Semantically Interoperable Systems. *12th International Conference on Information and Knowledge Management (CIKM'03)*, New Orleans, USA, November 2003, pp. 366–369

[113] T. Gruber. A Translation Approach to Portable Ontology Specification. *Knowledge Acquisition*, 5(2), 1993, pp. 199–220.

[114] T. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43(5-6) November-December 1995, pp. 907–928.

[115] D. Fensel, I. Horrocks, F. van Harmelen, D. McGuinness and P. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2), March-April 2001, pp. 38–45.

[116] N. Guarino and P. Giaretta. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (ed.), *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, IOS Press, Amsterdam, 1995, pp. 25–32.

[117] N. Guarino. Formal Ontology and Information Systems. *International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, June 1998, pp. 3–15.

[118] H. Chen, T. Finin and A. Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3), September 2003, pp. 197–207.

[119] N. Noy and D. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. *Technical Report KSL-01-05*, Knowledge Systems Laboratory, Stanford University, March 2001. Available at http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html; accessed December 2006.

[120] Foundation for Intelligent Physical Agents. FIPA Device Ontology Specification. *Specification number SI00091*, Geneva, Switzerland, December 2002. Available at http://www.fipa.org/specs/fipa00091/; accessed December 2006.

[121] Foundation for Intelligent Physical Agents. FIPA Quality of Service Specification. *Specification number SC00094*, Geneva, Switzerland, December 2002. Available at http://www.fipa.org/specs/fipa00094/; accessed December 2006.

[122] H. Helin and M. Laukkanen. Wireless Network Ontology. *Wireless World Research Forum 9th Meeting*, Zurich, Switzerland, July 2003.

[123] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. Butler and L. Tran (editors). Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. *W3C Recommendation 15 January 2004*. Available at http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/; accessed December 2006.

[124] R. Ocampo, A. Galis, C. Todd and H. De Meer. Towards Context-Based Flow Classification. *International Conference on Autonomous and Autonomic Systems (ICAS'06)*, July 2006, San Jose, California, USA, July 2006, pp. 44–53.

[125] R. Jasper and M. Uschold. A Framework for Understanding and Classifying Ontology Applications. *Proc. IJCAI-99 Workshop on Ontologies and Problem Solving Methods*, Stockholm, Sweden, August 1999. Available at http://CEUR-WS.org/Vol-18/; accessed December 2006.

[126] T. Berners-Lee. Semantic Web Road Map. September 1998. Available at http://www.w3.org/DesignIssues/Semantic.html; accessed December 2006.

[127] T. Berners-Lee, R. Fielding and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. *Request for Comments 2396*, August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt; accessed December 2006.

[128] T. Bray, D. Hollander and A. Layman. Namespaces in XML. *World Wide Web Consortium 14-January-1999.* Available at http://www.w3.org/TR/REC-xml-names/; accessed December 2006.

[129] F. Manola and E. Miller (editors). RDF Primer. *W3C Recommendation 10 February 2004.* Available at http://www.w3.org/TR/rdf-primer/; accessed December 2006.

[130] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation 10 February 2004.* Available at http://www.w3.org/TR/rdf-schema/; accessed December 2006.

[131] W3C Semantic Web Activity Statement. Available at http://www.w3.org/2001/sw/Activity.html; accessed December 2006.

[132] T. Berners-Lee. Semantic Web on XML. *Keynote presentation for XML 2000.* Slides available at http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html; accessed December 2006.

[133] M. Denny. Ontology Tools Survey, Revisited. *O'Reilly XML.com*, July 2004. Available at http://www.xml.com/pub/a/2004/07/14/onto.html; accessed December 2006.

[134] H. Knublauch, R. Ferguson, N. Noy and M. Musen. The Protégé OWL Plugin: An Open Environment for Semantic Web Applications. *Third International Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, November 2004, pp. 229–243.

[135] M. Horridge, H. Knublauch, A. Rector, R. Stevens and C. Wroe. A Practical Guide to Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools (Edition 1.0), August 2004. Available at http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf; accessed December 2006.

[136] Racer Systems GmbH & Co. KG. *RacerPro Reference Manual Version 1.9*, December 2005.

[137] International Organization for Standardization JTC 1. Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. *International Standard ISO/IEC 7498-1*, Reference number ISO/IEC 7498-1:1994(E), June 1996.

[138] H. Zimmermann. OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4), April 1980, pp. 425–432.

[139] F. Baker (ed.) Requirements for IP Version 4 Routers. *Request for Comments 1812*, June 1995. Available at http://www.ietf.org/rfc/rfc1812; accessed December 2006.

[140] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. *Request for Comments 4291*, February 2006. Available at http://www.ietf.org/rfc/rfc4291; accessed December 2006.

[141] P. Mockapetris. Domain Names - Concepts and Facilities. *Request for Comments 1034*, November 1987. Available at http://www.ietf.org/rfc/rfc1034; accessed December 2006.

[142] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 7(5), September 1993, pp. 8–18.

[143] E. Rosen, A. Viswanathan and R. Callon. Multiprotocol Label Switching Architecture. *Request for Comments 3031*, January 2001. Available at http://www.ietf.org/rfc/rfc3031; accessed December 2006.

[144] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, P. Larroy (editors). Linux Advanced Routing and Traffic Control. Available at http://www.lartc.org; accessed December 2006.

[145] H. Sofia Pinto, A. Perez and J. Martins. Some Issues on Ontology Integration. *Proc. IJCAI-99 Workshop on Ontologies and Problem-Solving Methods*, Stockholm, Sweden, August 1999. Available at http://CEUR-WS.org/Vol-18/; accessed December 2006.

[146] Foundation for Intelligent Physical Agents. FIPA Ontology Service Specification. *Specification number XC00086D*, Geneva, Switzerland, August 2001. Available at http://www.fipa.org/specs/fipa00086/; accessed December 2006.

[147] J. Quittek, S. Bryant, B. Claise and J. Meyer. Information Model for IP Flow Information Export. *Internet Draft (work in progress)*, June 2006. Available at http://tools.ietf.org/wg/ipfix/draft-ietf-ipfix-info/draft-ietf-ipfix-info-12.txt; accessed December 2006.

[148] Wireless Application Protocol Forum. WAP-174: UAProf User Agent Profiling Specification (1999), as amended by WAP-174_100 User Agent Profiling Specification Information Note (2001). Available at http://www.wapforum.org/what/technical_1_2.htm; accessed December 2006.

[149] Open Mobile Alliance. User Agent Profile V2.0 Base Vocabulary description, available at http://www.openmobilealliance.org/tech/profiles/ccppschema-20030226.html. The corresponding schema for this vocabulary may be found at http://www.openmobilealliance.org/tech/profiles/uaprof/ccppschema-20030226. Both URLs accessed December 2006.

[150] Open Mobile Alliance. User Agent Profile: Approved Version 2.0 – 06 Feb 2006. *Open Mobile Alliance OMA-TS-UAProf-V2_0-20060206-A.* Available at http://www.openmobilealliance.org/release_program/docs/UAProf/V2_0-20060206-A/OMA-TS-UAProf-V2_0-20060206-A.pdf; accessed December 2006.

[151] M. Dean and G. Schreiber (editors), S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider and L. Stein. OWL Web Ontology Language Reference. *W3C Recommendation 10 February 2004.* Available at http://www.w3.org/TR/owl-ref/; accessed December 2006.

[152] N. Noy (editor), M. Uschold and C. Welty (contributors). Representing Classes as Property Values on the Semantic Web. *W3C Working Group Note 5 April 2005.* Available at http://www.w3.org/TR/swbp-classes-as-values/; accessed December 2006.

[153] N. Noy and A. Rector (editors), P. Hayes and C. Welty (contributors). Defining N-ary Relations on the Semantic Web. *W3C Working Group Note 12 April 2006.* Available at http://www.w3.org/TR/swbp-n-aryRelations/; accessed December 2006.

[154] U. Hahn, S. Schulz and M. Romacker. Part-Whole Reasoning: A Case Study in Medical Ontology Engineering. *IEEE Intelligent Systems and their Applications*, 14(5). September 1999, pp. 59–67.

[155] A. Rector and C. Welty. Simple Part-Whole Relations in OWL Ontologies. *W3C Working Draft 15 Jan 2005.* Available at http://www.cs.man.ac.uk/~rector/swbp/simple-part-whole/simple-part-whole-relations-v0-2.html; accessed December 2006.

[156] H. Chen, F. Perich, T. Finin and A. Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. *International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous 2004)*, Boston, MA, USA, August 2004, pp. 258–267.

[157] D. Brickley and L. Miller. FOAF Vocabulary Specification. *Namespace Document 27 July 2005 - ('Pages about Things' Edition)*, latest revision dated January 2006. Available at http://xmlns.com/foaf/0.1/; accessed December 2006.

[158] F. Pan and J. Hobbs. Time in OWL-S. *Proceedings of the AAAI-04 Spring Symposium on Semantic Web Services*, Stanford, CA, USA, 2004, pp. 29–36.

[159] D. Lenat and R. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project.* Addison-Wesley, Reading, Massachusetts, USA, February 1990.

[160] D. Randell, Z. Cui and A. Cohn. A Spatial Logic Based on Regions and Connection. *Proc. 3rd International Conference on the Principles of Knowledge Representation and Reasoning (KR'92)*, Cambridge, MA, USA, October 1992, pp. 165–176.

[161] F. Perich (editor). MoGATU BDI Ontology. Available at http://mogatu.umbc.edu/bdi/; accessed December 2006.

[162] L. Kagal, T. Finin and A. Joshi. A Policy Based Approach to Security for the Semantic Web. *Proc. 2nd International Semantic Web Conference (ISWC2003)*, Florida, USA, October 2003, pp. 402–418.

[163] M. Laukkanen, H. Helin and H. Laamanen. Supporting Nomadic Agent-Based Applications in the FIPA Agent Architecture. *Proc. 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy, July 2002, pp. 1348–1355.

[164] H. Helin, H. Laamanen and M. Laukkanen. On Using Ontologies in Context Management. Available at https://bscw.ambient-networks.org/bscw/bscw.cgi/5126. (Accessed December 2006; accessible to Ambient Networks project partners only).

[165] H. Helin. *Supporting Nomadic Agent-Based Applications in the FIPA Architecture.* PhD thesis, Report A-2003-2, Department of Computer Science, University of Helsinki, Finland, February 2003. Available at http://ethesis.helsinki.fi/julkaisut/mat/tieto/vk/helin/supporti.pdf; accessed December 2006.

[166] A. Karlsson, R. Giaffreda, R. Glitho, J. Mattam, T. Kanter, C. Reichert, M. Smirnov, A. Karmouch, D. Balakrishnan, H. Harroud, H. Laamanen, R. Ocampo, L. Cheng, K. Jean, A. Galis and R. Lewis. Proof of Concept Demos - Role and Opportunities for Context Management in Ambient Networks. *Deliverable Report D-6-2, Ambient Networks Project*, July 2005. Document number IST-2002-507134-AN/WP6/D62. Available at http://www.ambient-networks.org/phase1web/publications/D_6_2.pdf; accessed December 2006.

[167] M. Khedr and A. Karmouch. Negotiating Context Information in Context-Aware Systems. *IEEE Intelligent Systems Magazine*, 19(6), November-December 2004, pp. 21–29.

[168] M. Khedr, M. Ganna, E. Horlait and A. Karmouch. Semantic Techniques for Reconfiguring and Adapting Networks in Pervasive Environments. *9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, Nice, France, May 2005, pp. 645–658.

[169] J. Lopez de Vergara, V. Villagra, J. Asensio and J. Berrocal. Ontologies: Giving Semantics to Network Management Models. *IEEE Network*, 17(3), May-June 2003, pp. 15–21.

[170] J. Lopez de Vergara, V. Villagra and J. Berrocal. Benefits of Using Ontologies in the Management of High Speed Networks. *7th IEEE International Conference on High Speed Networks and Multimedia Communications (HSNMC 2004)*, Toulouse, France, July 2004, pp. 1007-1018.

[171] J. Lopez de Vergara, V. Villagra and J Berrocal. Applying the Web Ontology Language to Management Information Definitions. *IEEE Communications Magazine*, 42(7), July 2004, pp. 68–74.

[172] Foundation for Intelligent Physical Agents. FIPA Nomadic Application Support Specification. *Specification number SI00014H*, Geneva, Switzerland, December 2002. Available at http://www.fipa.org/specs/fipa00014/; accessed December 2006.

[173] J. Martínez (editor). MPEG-7 Overview (version 10). ISO/IEC JTC1/SC29/WG11 N6828, Palma de Mallorca, October 2004. Available at http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm; accessed December 2006.

[174] C. Tsinaraki, P. Polydoros and S. Christodoulakis. Integration of OWL ontologies in MPEG-7 and TV-Anytime Compliant Semantic Indexing. *16th International Conference on Advanced Information Systems Engineering (CAiSE)*, Riga, Latvia, June 2004, pp. 398–413.

[175] C. Tsinaraki, P. Polydoros, N. Moumoutzis and S. Christodoulakis. Coupling OWL with MPEG-7 and TV-Anytime for Domain-Specific Multimedia Information Integration and Retrieval. *7th RIAO Conference 2004*, Avignon, France, April 2004. Available at http://www.music.tuc.gr/Staff/Director/Publications/publ_files /C_TPMC_RIAO_2004.pdf; accessed December 2006.

[176] A. Kalyanpur, B. Parsia, E. Sirin and J. Hendler. Debugging Unsatisfiable Classes in Owl Ontologies. *Journal of Web Semantics - Special Issue of the Semantic Web Track of WWW2005*, 3(4), 2005, pp. 268–293.

[177] Cisco Systems. *Introduction to Cisco IOS Netflow – A Technical Overview*, February 2006. Available at http://www.cisco.com/application /pdf/en/us/guest/products/ps6601/c1244/ cdccont_0900aecd80406232.pdf; accessed December 2006.

[178] P. Phaal and M. Lavine. sFlow Version 5. July 2004. Available at http://www.sflow.org/sflow_version_5.txt; accessed December 2006.

[179] J. Saltzer, D. Reed and D. Clark. End-to-End Arguments in System Design. *ACM Transactions in Computer Systems*, 2(4), November 1984, pp. 277–288.

[180] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based internets: MIB-II. *Request for Comments 1213*, March 1991. Available at http://www.ietf.org/rfc/rfc1213.txt; accessed December 2006.

[181] S. Waldbusser and P. Grillo. Host Resources MIB. *Request for Comments 2790*, March 2000. Available at http://www.ietf.org/rfc/rfc2790.txt; accessed December 2006.

[182] C. Estan, S. Savage and G. Varghese. Automatically Inferring Patterns of Resource Consumption in Network Traffic. *ACM SIGCOMM Conference 2003*, Karlsruhe, Germany, August 2003, pp. 137–148.

[183] J. Newsome, B. Karp and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. *IEEE Symposium on Security and Privacy*, May 2005, pp. 226–241.

[184] T. Karagiannis, K. Papagiannaki and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. *ACM SIGCOMM Conference 2005*, Philadelphia, PA, USA, 35(4), August 2005, pp. 229–240.

[185] R. Rehman. *Intrusion Detection with SNORT: Advanced IDS Techniques Using SNORT, Apache, MySQL, PHP, and ACID*, Prentice-Hall, May 2003.

[186] S. Kleene. Representation of Events in Nerve Nets and Finite Automata. *Automata Studies*, Princeton University Press, 1956.

[187] R. Nigel Horspool. Practical Fast Searching in Strings. *Software Practice and Experience*, 10(6), 1980, pp. 501–506.

[188] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente and D. Villela. A Survey of Programmable Networks. *ACM SIGCOMM Computer Communication Review*, 29(2), April 1999, pp. 7–23.

[189] J. Smith and S. Nettles. Active Networking: One View of the Past, Present, and Future. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(1), February 2004, pp. 4–18.

[190] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. *Computer Communication Review*, 26(2), April 1996, pp. 5–17.

[191] A. Ghosh, M. Fry and G. MacLarty. An Infrastructure for Application Level Active Networking. *Computer Networks*, 36(1), June 2001, pp. 5–20.

[192] A. Ghosh, M. Fry and J. Crowcroft. An Architecture for Application Layer Routing. *Proc. Second International Working Conference on Active Networks (IWAN 2000)*, Tokyo, Japan, October 2000, pp. 71–86.

[193] H. de Meer and P. O'Hanlon. Segmented Adaptation of Traffic Aggregates. *Proc. 9th International Workshop on Quality of Service (IWQoS 2001)*, Karlsruhe, Germany, June 2001, pp. 342–356.

[194] H. de Meer, K. Tutschku, and P. Tran-Gia. Dynamic Operation of Peer-to-Peer Overlay Networks. *Praxis der Informationsverarbeitung und Kommunikation (PIK) Journal, Special Issue on Peer-to-Peer Systems*, June 2003. Available at http://www3.informatik.uni-wuerzburg.de/papers/jour_13.pdf; accessed December 2006.

[195] D. Raz, A. Juhola, J. Serrat-Fernandez and A. Galis. *Fast and Efficient Context-Aware Services.* John Wiley and Sons, 2006.

[196] D. Raz and Y. Shavitt. An Active Network Approach for Efficient Network Management. *Proc. 1st International Working Conference on Active Networks 1999 (IWAN99)*, Berlin, Germany, July 1999, pp. 220–231.

[197] D. S. Alexander, B. Braden, C. Gunter, A. Jackson, A. Keromytis, G. Minden and D. Wetherall. The Active Network Encapsulation Protocol (ANEP), July 1997. Available at http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt; accessed December 2006.

[198] R. Presuhn (editor), J. Case, K. McCloghrie, M. Rose and R. Waldbusser. Management Information Base (MIB) for the Simple Network Management

Protocol (SNMP). *Request for Comments 3418*, December 2002. Available at http://www.ietf.org/rfc/rfc3418.txt; accessed December 2006.

[199] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss. An Architecture for Differentiated Services. *Request for Comments 2475*, December 1998. Available at http://www.ietf.org/rfc/rfc2475.txt; accessed December 2006.

[200] J. Hightower and G. Borriello. A Survey and Taxonomy of Location Systems for Ubiquitous Computing. *Technical Report UW-CSE 01-08-03*, Computer Science and Engineering, University of Washington, August 2001. Available at http://seattle.intel-research.net/people/jhightower/pubs/hightower2001survey/hightower2001survey.pdf; accessed December 2006.

[201] J. Navas and T. Imielinski. GeoCast - Geographic Addressing and Routing. *Proc. 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 1997)*, Budapest, Hungary, 1997, pp. 66–76.

[202] S. Basagni, I. Chlamtac, V. Syrotiuk and B. Woodward. A Distance Routing Effect Algorithm for Mobility (DREAM). *Proc. Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 1998)*, Dallas, TX, USA, October 1998, pp. 76–84.

[203] Y. Ko and N. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. *Wireless Networks*, 6(4), July 2000, pp. 307–321.

[204] B. Karp and H. Kung. Greedy Perimeter Stateless Routing for Wireless Networks. *Proc. 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, MA, USA, August 2000, pp. 243–254.

[205] P. Bahl and V. N. Padmanabhan. RADAR: An In-Building RF-Based User Location And Tracking System. *Proc. IEEE INFOCOM 2000*, Tel-Aviv, Israel, vol. 2, March 2000, pp. 775–784.

[206] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The Anatomy of a Context-Aware Application. *Proc. 5th Annual ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MOBICOM)*, Seattle, WA, USA, August 1999, pp. 59–68.

[207] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, Boston, MA, USA, August 2000, pp. 32–43.

[208] R. Klukas. *A Superresolution Based Cellular Positioning System Using GPS Time Synchronization.* PhD thesis, Department of Geomatics Engineering, University of Calgary, December 1997. Available at http://www.geomatics.ucalgary.ca/Papers/Thesis/GL/97.20114.RKlukas.pdf; accessed December 2006.

[209] B. Sklar. *Digital Communications: Fundamentals and Applications*, 2nd edition, Prentice-Hall, Englewood Cliffs, NJ, USA, 2000.

[210] L. Girod and D. Estrin. Robust Range Estimation Using Acoustic and Multimodal Sensing. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, Maui, Hawaii, USA, October 2001, pp. 1312–1320.

[211] R. Gold. Optimal Binary Sequences for Spread Spectrum Multiplexing. *IEEE Transactions on Information Theory*, IT-13(5), October 1967, pp. 619–621.

[212] R. Gold. Maximal Recursive Sequences for Spread Spectrum Multiplexing. *IEEE Transactions on Information Theory*, IT-14(1), January 1968, pp. 154–156.

[213] M. Roughan, S. Sen, O. Spatscheck and N. Duffield. Class-of-Service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification. *Proc. 4th ACM Internet Measurement Conference 2004 (IMC'04)*, Sicily, Italy, November 2004, pp. 135–148.

[214] S. Zander, T. Nguyen and G. Armitage. Self-Learning IP Traffic Classification Based on Statistical Flow Characteristics. *Proc. 6th International Workshop on Passive and Active Measurement (PAM 2005)*, Boston, MA, USA, March 2005, pp. 325–328.

[215] A. Moore and K. Papagiannaki. Toward the Accurate Identification of Network Applications. *Proc. 6th International Workshop on Passive and Active Measurement (PAM 2005)*, Boston, MA, USA, March 2005, pp. 41–54.

[216] H. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. *Proc. 13th USENIX Security Symposium*, San Diego, CA, USA, August 2004, pp. 271–286.

[217] C. Kreibich and J. Crowcroft. Honeycomb – Creating Intrusion Detection Signatures Using Honeypots. *Computer Communication Review*, 34(1), 2004, pp. 51–56.

[218] S. Singh, C. Estan, G. Varghese and S. Savage. Automated Worm Fingerprinting. *6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI'04)*, San Francisco, CA, USA, December 2004, pp. 45–60.

[219] H. Durrant-Whyte. Sensor Modules and Multisensor Integration. *International Journal of Robotics Research*, 7(6), December 1988, pp. 97–113.

[220] D. Hall and H. Llinas. An Introduction to Multisensor Data Fusion. *Proceedings of the IEEE*, 85(1), January 1997, pp. 6–23.

[221] M. Hazas and A. Ward. A High-Performance Privacy-Oriented Location System. *Proc. 1st IEEE International Conference on Pervasive Computing and Communications (PERCOM '03)*, Dallas-Fort Worth, USA, March 2003, pp. 216–223.

[222] USNO NAVSTAR Global Positioning System. http://tycho.usno.navy.mil/gpsinfo.html; accessed December 2006.

[223] E. Guttman, C. Perkins, J. Veizades and M. Day. Service Location Protocol, Version 2. *Request for Comments 2608*, June 1999. Available at http://www.ietf.org/rfc/rfc2608.txt; accessed December 2006.

[224] Sun Microsystems. *Jini Specifications Archive - v2.1*, 2005. Available at http://java.sun.com/products/jini/2_1index.html; accessed December 2006.

[225] UPnP Forum. *UPnP Device Architecture 1.0*, Version 1.0.1, December 2003, available at http://www.upnp.org/resources/upnpresources.zip; accessed December 2006.

[226] L. Clement, A. Hately, C. von Riegen and T. Rogers (editors). UDDI Version 3.0.2. *UDDI Spec Technical Committee Draft, Dated 20041019*, October 2004. Available at http://uddi.org/pubs/uddi_v3.htm; accessed December 2006.

[227] Salutation Consortium. *Salutation Architecture Specification Version 2.1*, 1999. (http://salutation.org unreachable as of December 2006.)

[228] C. Bettstetter and C. Renner. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. *EUNICE Open European Summer School*, Twente, The Netherlands, September 2000. Available at http://www.bettstetter.com/publications/bettstetter-2000-eunice-slp.pdf; accessed December 2006.

[229] A. Rakotonirainy and G. Groves. Resource Discovery for Pervasive Environments. *Proc. 4th International Symposium on Distributed Objects and Applications*, October 2002, pp. 866–883.

[230] R. Marin-Perianu, P. Hartel and H. Scholten. A Classification of Service Discovery Protocols. *Technical Report TR-CTIT-05-25*, Centre for Telematics and Information Technology, University of Twente, The Netherlands, June 2005. Available at http://eprints.eemcs.utwente.nl/735/; accessed December 2006.

[231] F. Zhu, M. Mutka and L. Ni. Service Discovery in Pervasive Computing Environments. *IEEE Pervasive Computing*, 4(4), October 2005, pp. 81–90.

[232] A. Mian, R. Beraldi and R. Baldoni. Survey of Service Discovery Protocols in Mobile Ad Hoc Networks. *Technical Report - Midlab 4/06*, Dip. Informatica e Sistemistica "Antonio Ruberti," Università degli Studi di Roma "La Sapienza," Rome, Italy, 2006. Available at http://www.dis.uniroma1.it/~midlab/articoli/SSDP.pdf; accessed December 2006.

[233] C. Reichert, M. Kleis and R. Giaffreda. Towards Distributed Context Management in Ambient Networks. *14th IST Mobile and Wireless Communications Summit*, Dresden, Germany, June 2005. Available at http://www.eurasip.org/content/Eusipco/IST05/papers/420.pdf; accessed December 2006.

[234] C. Sun, Y. Lin and B. Kemme. Comparison of UDDI Registry Replication Strategies. *IEEE International Conference on Web Services (ICWS'04)*, San Diego, CA, USA, July 2004, pp. 218–225.

[235] G. Manku. *Dipsea: A Modular Distributed Hash Table.* PhD thesis, Stanford University, August 2004. Available at http://infolab.stanford.edu/~manku/phd/thesis.pdf; accessed December 2006.

[236] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker. A Scalable Content-Addressable Network. *Proc. ACM SIGCOMM Conference 2001*, San Diego, CA, USA, August 2001, pp. 161–172.

[237] T. Cormen, C. Leiserson and R. Rivest. *Introduction to Algorithms*, MIT Press, 1990.

[238] N. Davies, S. Wade, A. Friday and G. Blair. Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications. *International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97)*, Toronto, Canada, May 1997, pp. 291–302.

[239] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. *IEEE/ACM Transactions on Networking (TON)*, 12(2), April 2004, pp. 205–218.

[240] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems (TOPLASS)*, 7(1), January 1985, pp. 80–112.

[241] L. Erman, F. Hayes-Roth, V. Lesser and D. Reddy. The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *ACM Computing Surveys*, 12(2), June 1980, pp. 213–253.

[242] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris and I. Stoica. Looking Up Data in P2P Systems. *Communications of the ACM*, 46(2), February 2003, pp. 43–48.

[243] R. Hancock, G. Karagiannis, J. Loughney and S. Van den Bosch. Next Steps in Signaling (NSIS): Framework. *Request for Comments 4080*, June 2005. Available at http://www.ietf.org/rfc/rfc4080.txt; accessed December 2006.

[244] M. Brunner (editor). Requirements for Signaling Protocols. *Request for Comments 3726*, April 2004. Available at http://www.ietf.org/rfc/rfc3726.txt; accessed December 2006.

[245] J. Postel. User Datagram Protocol. *Request for Comments 768*, August 1980. Available at http://www.ietf.org/rfc/rfc768.txt; accessed December 2006.

[246] D. Katz. IP Router Alert Option. *Request for Comments 2113*, February 1997. Available at http://www.ietf.org/rfc/rfc2113.txt; accessed December 2006.

[247] C. Partridge and A. Jackson. IPv6 Router Alert Option. *Request for Comments 2711*, October 1999. Available at http://www.ietf.org/rfc/rfc2711.txt; accessed December 2006.

[248] X. Deng, V. Haarslev and N. Shiri. A Framework for Explaining Reasoning in Description Logics. *Proc. International Symposium on Explanation-aware Computing, AAAI Fall Symposium 2005*, Washington DC, USA, November 2005. Available at http://www.cs.concordia.ca/~haarslev/publications/AAAI-FSS-2005.pdf; accessed December 2006.

[249] V. Haarslev and R. Möller. Racer: A Core Inference Engine for the Semantic Web. *Proc. 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, Florida, USA, October 2003, pp. 27–36.

[250] V. Haarslev and R. Möller. Practical Reasoning in RACER with a Concrete Domain for Linear Inequations. *Proc. International Workshop on Description Logics (DL-2002)*, Toulouse, France, April 2002, pp. 91–98.

[251] M. Wessel and R. Möller. A High Performance Semantic Web Query Answering Engine. *2005 International Workshop on Description Logics (DL2005)*, Edinburgh, Scotland, UK, July 2005. Available at http://www.sts.tu-harburg.de/~r.f.moeller/papers/2005/WeMo05a.pdf; accessed December 2006.

[252] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member Submission 21 May 2004*. Available at http://www.w3.org/Submission/SWRL/; accessed December 2006.

[253] I. Horrocks and P. Patel-Schneider. A Proposal for an OWL Rules Language. *Proc. 13th International Conference on World Wide Web (WWW 2004)*, NY, USA, 2004, pp. 723–731.

[254] R. Pang, M. Allman, V. Paxson and J. Lee. The Devil and Packet Trace Anonymization. *ACM Computer Communication Review*, 36(1), January 2006, pp. 29–38.

[255] Racer Systems GmbH & Co. KG. *RacerPro User's Guide Version 1.9*, December 2005.

[256] T. Chiueh, A. Neogi and P. Stirpe. Performance Analysis of an RSVP-Capable Router. *Proc. IEEE Real-Time Technology and Application Symposium*, Denver, Colorado, USA, June 1998, pp. 39-48.

[257] P. Pan and H. Schulzrinne. Processing Overhead Studies in Resource Reservation Protocols. *Proc. 17th. International Teletraffic Congress (ITC)*, Salvador, Brazil, September 2001. Available at http://www1.cs.columbia.edu/~pingpan/papers/lightweight.pdf; accessed December 2006.

[258] A. Farquhar, R. Fikes and J. Rice. Tools for Assembling Modular Ontologies in Ontolingua. *Technical Report KSL-97-03*, Knowledge Systems Laboratory, Stanford University, 1997. Available at ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-97-03.ps.gz; accessed December 2006.

[259] C. Perkins. IP Mobility Support. *Request for Comments 2002*, October 1996. Available at http://www.ietf.org/rfc/rfc2002.txt; accessed December 2006.

[260] C. Perkins and D. Johnson. Route Optimization in Mobile IP. *Internet Draft (work in progress)*, November 2000. Available at http://monarch.cs.cmu.edu/internet-drafts/draft-ietf-mobileip-optim-10.txt; accessed December 2006.

[261] M. Stemm and R. Katz. Vertical Handoffs in Wireless Overlay Networks. *Mobile Networks and Applications. Special Issue: Mobile Networking in the Internet*, 3(4), 1999, pp. 335–350.

[262] N. Shacham. Multipoint Communications by Hierarchically Encoded Data. *Proc. IEEE INFOCOM 1992*, Vol. 3, May 1992, pp. 2107–2114.

[263] D. Taubman and A. Zakhor. A Multi-Rate 3-D Subband Coding of Video. *IEEE Transactions on Image Processing*, 3(5), September 1994, pp. 572–588.

[264] D. Hoffman and M. Speer. Hierarchical Video Distribution over Internet-Style Networks. *Proc. IEEE International Conference on Image Processing*, Vol. 1, September 1996, pp. 5–8.

[265] S. McCanne, V. Jacobson and M. Vetterli. Receiver-Driven Layered Multicast. *Proc. ACM SIGCOMM '96 Conference*, August 1996, pp. 117–130.

[266] A. Helmy, M. Jaseemuddin and G. Bhaskara. Multicast-based Mobility: A Novel Architecture for Efficient Micro-Mobility. *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on All-IP Wireless Networks*, 22(4), May 2004, pp. 677–690.

[267] K. Nichols and B. Carpenter. Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification. *Request for Comments 3086*, April 2001. Available at http://www.ietf.org/rfc/rfc3086.txt; accessed December 2006.

[268] X. Chen and J. Heidemann. Detecting Early Worm Propagation through Packet Matching. *USC Information Sciences Institute Technical Report ISI-TR-2004-585*, February 2004. Available at http://www.isi.edu/~johnh/PAPERS/Chen04a.pdf; accessed December 2006.

[269] J. Postel. Internet Control Message Protocol. *Request for Comments 792*, September 1981. Available at http://www.ietf.org/rfc/rfc792.txt; accessed December 2006.

[270] R. Twining, M. Williamson, M. Mowbray and M. Rahmouni. Email Prioritization: Reducing Delays on Legitimate Mail Caused by Junk Mail. *Proc. 2004 USENIX Annual Technical Conference*, Boston, MA, USA, June 2004, pp. 45–58.

[271] L. Donnerhacke. Teergrubing FAQ. Available at http://www.iks-jena.de/mitarb/lutz/usenet/teergrube.en.html; accessed December 2006.

[272] K. Li, C. Pu and M. Ahamad. Resisting Spam Delivery by TCP Damping. *1st Conference on Email and Anti-Spam (CEAS 2004)*, Mountain View, CA, USA, July 2004. Available at http://www.ceas.cc/papers-2004/191.pdf; accessed December 2006.

[273] V. Jacobson. Congestion Avoidance and Control. *Proceedings of SIGCOMM'88*, Palo Alto, CA, USA, August 1988, pp. 314–329.

[274] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1993, pp. 397–413.

[275] R. Clayton. Stopping Spam by Extrusion Detection. *1st Conference on Email and Anti-Spam (CEAS 2004)*, Mountain View, CA, USA, July 2004. Available at http://www.ceas.cc/papers-2004/172.pdf; accessed December 2006.

[276] D. Moore. C. Shannon, G. Voelker and S. Savage. Network Telescopes: Technical Report. *Technical report CS2004-0795*, UCSD Computer Science and Engineering, July 2004. Available at http://www.caida.org/publications/papers/2004/tr-2004-04/tr-2004-04.pdf; accessed December 2006.

[277] D. Reed, J. Saltzer, and D. Clark. Comment on Active Networking and End-to-End Arguments. *IEEE Network*, 12(3), May-June 1998, pp. 69–71.

[278] R. Boyer and J. Moore. A Fast String Searching Algorithm. *Communications of the ACM*, 20(10), October 1977, pp. 762–772.

[279] K. Lim and R. Stadler. Real-Time Views of Network Traffic Using Decentralised Management. *9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, Nice, France, May 2005, pp. 119–132.

[280] M. Castro, P. Druschel, Y. Hu and A. Rowstron. Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks. *Proc. International Workshop on Future Directions in Distributed Computing (FuDiCo)*, Bertinoro, Italy, June 2002, pp. 103–107.

[281] S. Ratnasamy, M. Handley, R. Karp and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. *Proc. IEEE INFOCOM 2002*, Vol. 3, NY, USA, June 2002, pp. 1190–1199.

[282] J. Newsome, B. Karp and D. Song. Paragraph: Thwarting Signature Learning by Training Maliciously. *9th International Symposium on Recent Advances in Intrusion Detection (RAID 2006)*, September 2006, pp. 81–105.

[283] R. Giaffreda, H. Tschofenig, T. Kanter and C. Reichert. An Authorisation and Privacy Framework for Context-Aware Networks. *2nd International Workshop on Mobility Aware Technologies and Applications (MATA 2005)*, Montreal, Canada, October 2005, pp. 149–160.

[284] R. Wishart, K. Henricksen and J. Indulska. Context Obfuscation for Privacy via Ontological Descriptions. *1st International Workshop on Location- and Context-Awareness (LoCA 2005)*, Germany, May 2005, pp. 276–288.

[285] T. Raita. Tuning the Boyer-Moore-Horspool String Searching Algorithm. *Software Practice and Experience*, 22(10), October 1992, pp. 879–884.

[286] W. Foy. Position-Location Solution by Taylor-Series Estimation. *IEEE Transactions on Aerospace and Electrical Systems (AES)*, 12(2), March 1976, pp. 187–193.

[287] G. Strang. *Linear Algebra and its Applications.* Thomson Learning, 1988.

[288] International Organization for Standardization JTC 1/SC34. Information Processing – Text and office systems – Standard Generalized Markup Language (SGML). *ISO Standard 8879*, 1986.

[289] D. Fallside and R. Walmsley. XML Schema Part 0: Primer Second Edition. *W3C Recommendation 28 October 2004*. Available at http://www.w3.org/TR/xmlschema-0/; accessed December 2006.

[290] G. Klyne and J. Carroll (editors). Resource Description Framework (RDF): Concepts and Abstract Syntax. *W3C Recommendation 10 February 2004*. Available at http://www.w3.org/TR/rdf-concepts/; accessed December 2006.

[291] D. McGuinness and F. van Harmelen (editors). OWL Web Ontology Language Overview. *W3C Recommendation 10 February 2004*. Available at http://www.w3.org/TR/owl-features/; accessed December 2006.

[292] G. Antoniou and F. van Harmelen. Web Ontology Language: OWL. In S. Staab and R. Studer (editors), *Handbook on Ontologies in Information Systems*, Springer-Verlag, 2003, pp. 76–92.

[293] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten and J-C. Burgelman. Scenarios for Ambient Intelligence in 2010. Final report of the IST Advisory Group, February 2001. Available at ftp://ftp.cordis.lu/pub/ist/docs/istagscenarios2010.pdf; accessed December 2006.

[294] B. Busropan, F. Pittman, S. Ruffino, O. Karasti, J. Gebert, T. Koshimizu, D. Moro, B. Ohlman, N. Papadoglou, A. Scheider, W. Speltacker, V. Typpö and S. Svaet. Ambient Network Scenario, Requirements and Draft Concepts. *Deliverable Report D-1-2, Ambient Networks Project*, October 2004. Document number IST-2002-507134-AN/WP1/D02. Available at http://www.ambient-networks.org/phase1web/publications/AN-D1-2%20public.pdf; accessed December 2006.

Appendix A

List of Publications

The following papers are relevant to the work described here and were written or co-written by the author of this thesis.

Peer-reviewed conferences, workshops and symposia

1. **R. Ocampo** and H. De Meer. A Method for Sensing and Representing Location in Context-Aware Applications. *London Communications Symposium*, London, UK, September 2003, pp. 233–266.

2. **R. Ocampo** and H. De Meer. Smart Wireless Access Points for Pervasive Computing. *First Working Conference on Wireless On-Demand Network Systems (WONS '04)*, Trento, Italy, January 2004, pp. 329–343.

3. **R. Ocampo**, A. Galis and C. Todd. Triggering Network Services Through Context-Tagged Flows. *2nd International Workshop on Active and Programmable Grids Architectures and Components (APGAC'05) / International Conference on Computational Science 2005 (ICCS 2005)*, Atlanta, GA, USA, May 2005, pp. 259–266.

4. **R. Ocampo**, A. Galis, H. De Meer and C. Todd. Implicit Flow QoS Signaling Using Semantic-Rich Context Tags. *13th International Workship on Quality of Service (IWQoS 2005)*, Passau, Germany, June 2005, pp. 369–371.

5. **R. Ocampo**, L. Cheng, Z. Lai and A. Galis. ContextWare Support for Network and Service Composition and Self-Adaptation. *2nd International Workshop on Mobility Aware Technologies and Applications (MATA 2005)*, Montreal, Canada, October 2005, pp. 84–95.

6. **R. Ocampo**, A. Galis, H. De Meer and C. Todd. Supporting Mobility Adaptation Through Flow Context. Short paper proceedings of the *2nd International Workshop on Mobility Aware Technologies and Applications (MATA 2005)*, Montreal, Canada, October 2005, pp. 16–20.

7. L. Cheng, **R. Ocampo**, A. Galis, R. Szabo, C. Simon, and P. Kersch. Self-Management in Ambient Networks for Service Composition. *2005 IFIP International Conference on Intelligence in Communication Systems (Intellcom 2005)*, Montreal, Canada, October 2005.

8. **R. Ocampo**, A. Galis, H. De Meer and C. Todd. Flow Context Tags: Concepts and Applications. *IFIP TC6 Conference on Network Control and Engineering for QoS, Security and Mobility (NetCon'05)*, Lannion, France, November 2005.

9. **R. Ocampo**, A. Galis, H. De Meer and C. Todd. Towards Context-Based Flow Classification. *International Conference on Autonomous and Autonomic Systems (ICAS'06)*, July 2006, San Jose, CA, USA, pp. 47–53.

10. L. Cheng, K. Jean, **R. Ocampo** and A. Galis. Service-Aware Overlay Adaptation in Ambient Networks. *International Multi-Conference on Computing in the Global Information Technology*, August 2006, Bucharest, Romania.

11. K. Jean, L. Cheng, **R. Ocampo** and A. Galis. Contextualisation of Management Overlays in Ambient Networks. *International Multi-Conference on Computing in the Global Information Technology*, August 2006, Bucharest, Romania.

12. L. Cheng, **R. Ocampo**, K. Jean, A. Galis and B. Lai. Towards Dynamic Distributed Hash Table Composition and Decomposition in Ambient Networks. *17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2006)*, October 2006, Dublin, Ireland, pp. 258–268.

13. **R. Ocampo**, L. Cheng, K. Jean, A. Gonzalez Prieto, A. Galis and Z. Lai. Towards a Context Monitoring System for Ambient Networks. *First International Conference on Communications and Networking in China (CHINACOM 2006)*, October 2006, Beijing, China.

Other technical papers and reports

1. R. Giaffreda, M. Barachi, R. Glitho, A. Jonsson, M. Smirnov, M. Kleis, A. Karmouch, M. Khedr, A. Karlsson, H. Laamanen, H. Helin, A. Galis, **R. Ocampo** and J. Zhang. Context Information Sources and Destinations and Relationship to AN Activities.

Internal Report R-6-2, Ambient Networks Project, October 2004. Document number IST-2002-507134-AN/WP6/R6-2. Available at https://bscw.ambient-networks.org/bscw/bscw.cgi/0/7751; accessible to Ambient Networks project partners only.

2. A. Jonsson, R. Giaffreda, M. Barachi, R. Glitho, F. Belqasmi, M. Smirnov, M. Kleis, C. Reichert, A. Karmouch, M. Khedr, A. Karlsson, H. Laamanen, H. Helin, A. Galis, **R. Ocampo** and J. Zhang. Ambient Networks ContextWare: First Paper on Context-Aware Networks. *Deliverable Report D-6-1, Ambient Networks Project*, January 2005. Document number IST-2002-507134-AN/WP6/D61, available at http://www.ambient-networks.org/phase1web/publications/D6-1_PU.pdf

3. A. Karlsson, R. Giaffreda, R. Glitho, J. Mattam, T. Kanter, C. Reichert, M. Smirnov, A. Karmouch, D. Balakrishnan, H. Harroud, H. Laamanen, **R. Ocampo**, L. Cheng, K. Jean, A. Galis and R. Lewis. Proof of Concept Demos – Role and Opportunities for Context Management in Ambient Networks. *Deliverable Report D-6-2, Ambient Networks Project*, July 2005. Document number IST-2002-507134-AN/WP6/D62. http://www.ambient-networks.org/phase1web/publications/D_6_2.pdf

4. M. Brunner, S. Schuetz, G. Nunzi, E. A. Asmare, K. Zimmermann, J. Nielsen, A. Gunnar, H. Abrahamsson, R. Szabo, S. Csaba, M. Erdei, P. Kersch, Z. L. Kis, B. Kovács, R. Stadler, A. Wágner, K. Molnár, A. Gonzalez, G. Molnar, J. Andres, M. Á. Callejo, L. Cheng, **R. Ocampo** and A. Galis. Ambient Network Management – Solution Design and Functions. *Deliverable Report D-8-2, Ambient Networks Project*, July 2005. Document number IST-2002-507134-AN/D8-2. Available at http://www.ambient-networks.org/phase1web/publications/D8-2_PU.pdf

5. **R. Ocampo**. Flow Context Tagging: Concepts and Applications. Unpublished transfer report, Department of Electronic and Electrical Engineering, University College London, July 2005.

6. R. Giaffreda, J. Dang, R. Glitho, M. E. Barachi, F. Belqasmi, J. Mattam, T. Kanter, C. Reichert, M. Smirnov, A. Karmouch, D. Balakrishnan, H. Harroud, A. Karlsson, H. Laamanen, M. Laukkanen, **R. Ocampo**, K. Jean and A Galis. Ambient Networks ContextWare: Second Paper on Context-Aware Networks. *Deliverable Report D-6-3, Ambient Networks Project*, December 2005. Document number IST-2002-507134-AN/WP6/D63. Available at http://www.ambient-networks.org/phase1web/ publications/ D6_3_Ambient_Networks_ ContextWare_Second_Paper_on_Context-Aware_Networks_PU.pdf

7. S. Schuetz, G. Nunzi, E. A. Asmare, K. Zimmermann, J. Nielsen, A. Gunnar, H. Abrahamsson, R. Szabo, S. Csaba, M. Erdei, P. Kersch, Z. L. Kis, B. Kovács, R.

Stadler, A. Wágner, K. Molnár, A. Gonzalez, F. Wuhib, G. Molnar, J. Andres, M. Á. Callejo, L. Cheng, **R. Ocampo** and A. Galis. Ambient Network Management – Proof-of-Concepts and Evaluations. *Deliverable Report D-8-3, Ambient Networks Project*, December 2005. Document number IST-2002-507134-AN/D8-3. Available at http://www.ambient-networks.org/phase1web/publications/ D8_3_ Ambient_Network_Management_Proof_of_Concepts_&_Evaluations_PU.pdf

8. A. Galis, F. Belqasmi, L. Cheng, S. Csaba, R. Giaffreda, R. Glitho, A. Gonzalez, H. Harroud, I. Herwono, K. Jean, T. Kanter, A. Karmouch, P. Kersch, Z. Kis, J. Nielsen, G. Nunzi, B. Ohlman, **R. Ocampo**, M. Rónai, N. Samaan and T. Szydło. Interim Report on Integrated Design for Context, Network and Policy Management. *Deliverable Report D10-R1, Ambient Networks Phase 2 Project*, July 2006. Document number FP6-CALL4-027662-AN P2/ D10-R1. Available at https://bscw.ambient-networks.org/bscw/bscw.cgi/375766; accessible to Ambient Networks project partners only.

Appendix B

The Boyer-Moore-Horspool Algorithm

The Boyer-Moore-Horspool (BMH) algorithm is a simplification by Horspool of the Boyer-Moore *exact string matching* algorithm [187, 278].

Assume that the payload of a PDU, `payload`, of length `payloadLength`, will be searched for the occurrence of a byte sequence `pattern`, of length `patternLength`. As is the case with most string search algorithms, a section of `payload` is compared character-by-character (bytewise in this case) with `pattern`. However, BMH does this by starting with the rightmost byte in `pattern` and comparing it with a corresponding byte position in `payload`, and progressively comparing each byte in `pattern` with the corresponding byte in `payload`.

When a mismatch occurs during the byte comparisons between `pattern` and `payload`, BMH shifts the entire pattern to the right to continue to search for an occurrence of `pattern` at another location within `payload`. However, while a "brute force" (BF) algorithm might simply shift `pattern` to the right by one position, BMH gets its performance advantage over BF by carefully computing the number of positions to the right it may shift, and may often end up shifting by more than one position. As illustrated in Figure B.1(a), `pattern` is compared byte-by-byte with a substring of `payload`, starting from the rightmost byte of `pattern`. Assume that after a series of matches, a mismatch is found at the position where `pattern` contains y and `payload` contains x. If `pattern` also contains x, then it is shifted such that the rightmost occurrence of x in `pattern` aligns with the corresponding x in `payload` (which caused the earlier mismatch), as shown in Figure B.1(b). Otherwise, if `pattern` does not contain any x, then it is shifted one position beyond the occurrence of x in `payload`, as in Figure B.1(c). It is easy to see the logic behind this

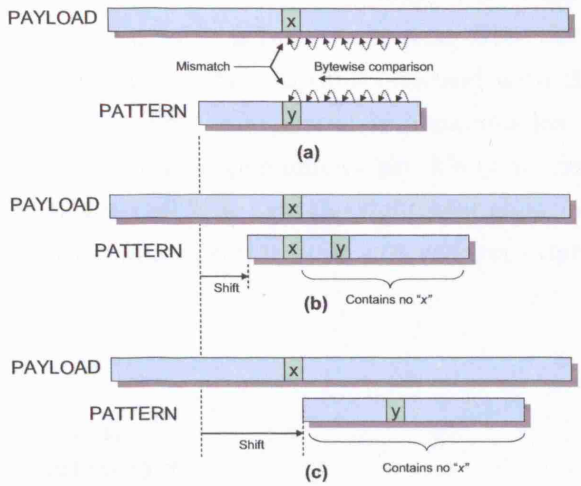


Figure B.1: The Boyer-Moore-Horspool algorithm.

latter case because if x is not contained in `pattern`, then it would be useless to try and find a bytewise match between `pattern` and any contiguous subset of bytes in `payload` that contains x.

Rather than dynamically compute the number of steps `pattern` has to be shifted in the event of a mismatch, BMH uses a "pre-compiled" lookup table (called $delta_{12}$ in [187]) that provides a precomputed shift value for each character in the search alphabet (or for each byte value in this case). Most of the entries in `delta12` are likely to have values equal to `patternLength`, and the only entries that would have values less than this would correspond to the individual characters (bytes) in `pattern`. The following Java code fragment from `FlowSensor` illustrates how `delta12` is pre-compiled:

```
private void precompile(byte[] pattern, int[] delta12) {
   int patternLength=pattern.length;
   //Initialize with default shift amount
   for(int i=0; i<delta12.length; i++) {
      delta12[i]=patternLength;
   }
   //Compute shift amount for each pattern symbol
   for(int i=0; i<patternLength-1; i++) {
      delta12[((int) pattern[i]) & 0xFF] = patternLength-i-1;
   }
}
```

Raita's variation After pre-compilation of the lookup table, the main part of BMH is executed using the Java code fragment below. Note that after the rightmost position in `pattern` has been matched, the algorithm continues instead with the rest of the bytewise comparisons from left to right. This is a partial implementation of a BMH variant by Raita, who argues that since strong dependencies are likely to exist between successive symbols, then it would be best to initially test the rightmost element in `pattern` (as in the usual case), then the *leftmost* element, rather than proceed from right to left as is normally done [285].

```
byte b;
int j = fromPosition;
while (j <= toPosition) {
   //Check if rightmost byte in pattern is matched
   b = payload[j+patternLength-1];
   if(pattern[patternLength-1] == b) {
      //Byte in payload matches last byte of pattern
      //Scan for matches in rest of pattern, from left to right
      for(int i=0; (i<patternLength-1) &&
         (pattern[i]==payload[i+j]); i++) {
         if (i >= patternLength-2) { //Matched all?
            return(j);               //Yes, return position
         }
      }
   }
   // Found a mismatch. Use the delta12 table
   // to determine the shift amount
   j = j + delta12[(int)b & 0xFF];
}
return(-1);
```

`delta12` is used in the penultimate line of the code fragment to determine the amount of shift (represented by the increment in the variable `j`) in the event of a mismatch. The fragment returns either -1 if no match was found in the entire payload, or the value of `j` representing the position in the payload where a match was found.

Appendix C

Hyperbolic Multilateration

For a receiver located at coordinates (x, y, z), and any pair $k = \{i, j\}$ of beacons i and j, located at (x_i, y_i, z_i) and (x_j, y_j, z_j) respectively, the equation describing the range difference r_k corresponding to the *time-difference-of-arrival* (TDOA) $t_i - t_j$ for this pair is given by:

$$\begin{aligned} r_k(x, y, z) &= c(t_i - t_j) \\ &= \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \\ &\quad -\sqrt{(x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2} \end{aligned} \tag{C.1}$$

where c is the propagation speed of the beacon signal used. Equation (C.1) is a hyperboloid with foci at the beacon positions (x_i, y_i, z_i) and (x_j, y_j, z_j), with points on its graph specifying possible locations of the receiver. This is illustrated for the two-dimensional case in Figure C.1.

With four beacon signals, three independent TDOA values may be obtained, producing three independent equations in the form of Equation (C.1). The solution to these three simultaneous equations yields a (x, y, z) position estimate of the receiver. Graphically, this corresponds to the intersection between the hyperboloids generated using Equation (C.1) for any set of three TDOA values.

The three simultaneous nonlinear equations in the form of Equation (C.1) can be solved by first linearizing them using their Taylor series approximations, and then by iteratively computing for estimates of the solution until some error criterion is met [286]. Performing

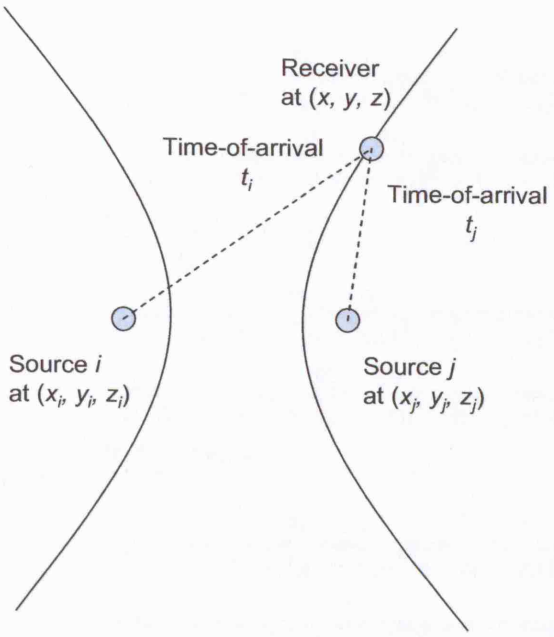


Figure C.1: Hyperbolic multilateration with two beacons

a Taylor series expansion on Equation (C.1) evaluated at point (x_0, y_0, z_0), and retaining the first two terms to obtain its first-order approximation $\hat{r_k}(x, y, z)$,

$$\begin{aligned}\hat{r_k}(x,y,z) &= r_k(x_0,y_0,z_0) + (x-x_0)\frac{\partial r_k(x_0,y_0,z_0)}{\partial x} \\ &+ (y-y_0)\frac{\partial r_k(x_0,y_0,z_0)}{\partial y} + (z-z_0)\frac{\partial r_k(x_0,y_0,z_0)}{\partial z} \\ &= r_k(x_0,y_0,z_0) + a_{kx}\delta_x + a_{ky}\delta_y + a_{kz}\delta_z\end{aligned} \tag{C.2}$$

where

$$\begin{aligned}r_k(x_0,y_0,z_0) &= \sqrt{(x_i-x_0)^2+(y_i-y_0)^2+(z_i-z_0)^2} \\ &\quad -\sqrt{(x_j-x_0)^2+(y_j-y_0)^2+(z_j-z_0)^2}\end{aligned}$$

$$
\begin{aligned}
a_{kx} &= \frac{\partial r_k(x_0, y_0, z_0)}{\partial x} \\
&= \frac{x_i - x_0}{\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2}} \\
&\quad - \frac{x_j - x_0}{\sqrt{(x_j - x_0)^2 + (y_j - y_0)^2 + (z_j - z_0)^2}} \\
a_{ky} &= \frac{\partial r_k(x_0, y_0, z_0)}{\partial y} \\
&= \frac{y_i - y_0}{\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2}} \\
&\quad - \frac{y_j - y_0}{\sqrt{(x_j - x_0)^2 + (y_j - y_0)^2 + (z_j - z_0)^2}} \\
a_{kz} &= \frac{\partial r_k(x_0, y_0, z_0)}{\partial z} \\
&= \frac{z_i - z_0}{\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2}} \\
&\quad - \frac{z_j - z_0}{\sqrt{(x_j - x_0)^2 + (y_j - y_0)^2 + (z_j - z_0)^2}}
\end{aligned}
$$

and $\delta_x = x - x_0$, $\delta_y = y - y_0$, $\delta_z = z - z_0$. Noting that each $k = 1, 2, 3 \ldots$ denotes a beacon-pair $\{i, j\}$, only three beacon pairs are selected to form three independent TDOA equations. Transposing the terms in Equation (C.2) and writing out the three simultaneous TDOA equations yields

$$a_{1x}\delta_x + a_{1y}\delta_y + a_{1z}\delta_z = \hat{r_1}(x, y, z) - r_1(x_0, y_0, z_0) \quad \text{(C.3)}$$

$$a_{2x}\delta_x + a_{2y}\delta_y + a_{2z}\delta_z = \hat{r_2}(x, y, z) - r_2(x_0, y_0, z_0) \quad \text{(C.4)}$$

$$a_{3x}\delta_x + a_{3y}\delta_y + a_{3z}\delta_z = \hat{r_3}(x, y, z) - r_3(x_0, y_0, z_0) \quad \text{(C.5)}$$

The three simultaneous equations may then be conveniently expressed in matrix form as

$$\mathbf{A}\delta = \mathbf{r} \quad \text{(C.6)}$$

where

$$\mathbf{A} = \begin{bmatrix} a_{1x} & a_{1y} & a_{1z} \\ a_{2x} & a_{2y} & a_{2z} \\ a_{2x} & a_{2y} & a_{2z} \end{bmatrix}, \quad \delta = \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_z \end{bmatrix}, \quad \mathbf{r} = \begin{bmatrix} \hat{r_1}(x,y,z) - r_1(x_0,y_0,z_0) \\ \hat{r_2}(x,y,z) - r_2(x_0,y_0,z_0) \\ \hat{r_3}(x,y,z) - r_3(x_0,y_0,z_0) \end{bmatrix}$$

A least squares solution to Equation (C.6) is given by [287]:

$$\hat{\delta} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{r} \tag{C.7}$$

where

$$\hat{\delta} = \begin{bmatrix} \hat{\delta_x} \\ \hat{\delta_y} \\ \hat{\delta_z} \end{bmatrix}$$

This is then used to produce new estimates from the initial guess (x_0, y_0, z_0):

$$\hat{x} = x_0 - \hat{\delta_x}$$
$$\hat{y} = y_0 - \hat{\delta_y}$$
$$\hat{z} = z_0 - \hat{\delta_z}$$

The process iterates until it either converges, i.e. changes in successive estimates are below a certain tolerance ϵ, or a maximum number of allowed iterations is exceeded. Note that $\hat{r_k}(x,y,z)$ is an approximation to $r_k(x,y,z)$, the actual position of the object being tracked. In Equation (C.7) the values for $r_k(x_0,y_0,z_0)$, representing the range difference between the guess point (x_0,y_0,z_0) and a pair of beacons $k = \{i,j\}$, are obtained in a straightforward way by evaluating Equation (C.1) at (x_0,y_0,z_0). On the other hand, the values for $\hat{r_k}(x,y,z) \approx r_k(x,y,z)$ are obtained from the TDOA readings using signals from the beacon pair $k = \{i,j\}$.

Appendix D

Overview of Semantic Web Technologies

This section provides an overview of the following technologies for the Semantic Web:

- *XML*, which provides a basic syntax for structured documents, but which provides no facility to encode the meaning of these documents.
- *XML Schema*, a language which specifies restrictions on the structure of XML documents
- *RDF*, which provides a data model for objects ("resources") and relations between them
- *RDF Schema*, a vocabulary for describing properties and classes of RDF resources, as well as support for hierarchies of such properties and classes, and
- *OWL*, which adds a richer vocabulary and formal semantics for describing properties, classes, and their relations.

Extensible Markup Language (XML)

The Extensible Markup Language, or XML [100], is a markup language that defines a scheme for logically structuring information within documents, allowing their exchange between applications. Each XML document contains one or more *elements* delimited by *tags*, which are either start-tags, end-tags, or, for empty elements, empty-element tags.

Each element has a type, identified by name, and may have a set of attribute specifications, each containing a name and a value.

Tags typically consist of data delimited by '<' and '>.' A simple example of a fragment of an XML document might be as follows:

```
<?xml version="1.0"?>
<hardwarePlatform>
    <cpu>Pentium III</cpu>
    <memory size="1024" unit="Mbytes">
    </memory>
    <screen>
        <color>yes</color>
        <screenSize>640x480</screenSize>
        <bitsPerPixel>24</bitsPerPixel>
    </screen>
    <soundOutputCapable>yes</soundOutputCapable>
</hardwarePlatform>
```

It is easy to see how XML provides structure to the information contained in the previous example, by providing nested sets of named elements. In the example, the document provides a description for some hardware platform, listing some information about its CPU, memory, screen and audio system. Within the screen element, one is able to see further detail about its size, color characteristics, and bit-resolution.

XML's syntax (refer [100] for the full and formal specification) is a subset of existing, widely used international text processing standard called the Standard Generalized Markup Language (SGML) [288], and documents that comply with XML's syntax are called *well-formed.*

XML Namespaces XML documents being used by multiple applications may face problems of having the same markup recognized by different applications, or for the same element types or attribute names in different documents to "collide" (be in conflict) when being processed by the same software module. A simple example of such a conflict is illustrated by the two examples below. While the `flow` element on the left seems to refer to a network flow, the one on the right seems to refer to a fluid flow.

```
<flow>
    <rate>10 Mbps</rate>
</flow>
```

```
<flow>
    <rate>10 lbs/min</rate>
</flow>
```

To resolve this, *XML namespaces* provide document constructs with universal names whose scope extend beyond the documents themselves [128]. The W3C defines an XML namespace as a collection of names, identified by a URI reference [127], which are used in XML documents as element types and attribute names. This URI reference is called the *namespace name*. The URI should have the characteristics of uniqueness and persistence, as its goal is to give the namespace a unique name.

An XML namespace attribute allows namespace *prefixes* to be defined and associated with namespace names. A prefix functions only as a placeholder for the namespace name (the URI) and is fully expanded by applications when parsing and constructing names within documents. Using the previous example,

```
<h:flow xmlns:h="http://ee.ucl.ac.uk/ComputerNetworks/">
    <h:rate>10 Mbps</h:rate>
</h:flow>
```

uses the `flow` and `rate` terms found in the `http://ee.ucl.ac.uk/ComputerNetworks/` namespace, while

```
<f:flow xmlns:f="http://ee.ucl.ac.uk/Hydraulics/">
    <f:rate>10 lbs/min</f:rate>
</f:flow>
```

refers to terms in the `http://ee.ucl.ac.uk/Hydraulics/` namespace.

XML Schema The XML Schema definition language [289] enables one to predefine the structure and vocabulary of a set of XML documents, allowing applications to define as *valid* those documents that conform to the specified syntax and structure. For example, the schema fragment below

```
<xsd:complexType name="flow">
    <xsd:sequence>
        <xsd:element name="flowID" type="xsd:string"
            minOccurs="1" maxOccurs="1"/>
        <xsd:element name="currentRate" type="xsd:double"
            minOccurs="0" maxOccurs="1"/>
        <xsd:element name="hop" type="xsd:string"
            minOccurs="0" maxOccurs="unbounded"/>
```

```
    </xsd:sequence>
  </xsd:complexType>
```

specifies an element `flow`, which is a `complexType`, that is, it contains other elements. The subelements are `flowID`, of type `string` that occurs exactly once; `currentRate`, of type `double`, which may or may not be present; and zero or more occurrences of the string `hop`.

Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web [129]. The underlying structure in RDF is the *triple*, which consists of a subject, a predicate or property, and an object. Each triple may be graphically depicted by two nodes (the subject and the object) linked by a directed arc (the predicate or property) between the two. The idea is that the triple asserts or expresses that the relationship exists between the things denoted by the subject and the object. A node may be a *URI reference* which is basically a URI with an optional "#" and fragment identifier at the end, a literal, or blank; while properties are URI references. A set of triples is an RDF graph, and the meaning of an RDF graph is the logical conjunction (logical AND) of all of the statements corresponding to its triples [290]. Although the conceptual model for RDF is a graph, to represent statements in a machine-processable way, RDF provides an XML syntax for representing and exchanging graphs, called *RDF/XML* [104].

RDF Schema Although RDF can specify properties either as attributes of resources, or relationships between resources, by itself it has no mechanisms for describing these properties or relationships [130]. *RDF Schema*, RDF's vocabulary description language, defines classes and properties that may be used to describe classes, properties and resources.

- *Classes* denote groups of resources, and `rdfs:Class` pertains to groups of resources that fall under RDF Schema classes. Members or *instances* of `rdfs:Class` include `rdfs:Resource`, which pertains to the class of all things described by RDF, and `rdf:Property`, the class of all RDF properties. A full list of RDF classes is given in Table D.1.
- *Properties* are defined as relations between subject and object resources [290]. Instances of `rdf:Property` include `rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, and `rdfs:subPropertyOf`. Resources that have a given property are said to belong to its `rdfs:domain`, while the values of the property belong to its `rdfs:range`. If all

Class name	Comment
rdfs:Resource	The class resource, everything.
rdfs:Literal	The class of literal values, e.g. textual strings and integers.
rdf:XMLLiteral	The class of XML literals values.
rdfs:Class	The class of classes.
rdf:Property	The class of RDF properties.
rdfs:Datatype	The class of RDF datatypes.
rdf:Statement	The class of RDF statements.
rdf:Bag	The class of unordered containers.
rdf:Seq	The class of ordered containers.
rdf:Alt	The class of containers of alternatives.
rdfs:Container	The class of RDF containers.
rdfs:ContainerMembershipProperty	The class of container membership properties, rdf:_1, rdf:_2, ..., all of which are sub-properties of 'member'.
rdf:List	The class of RDF Lists.

Table D.1: RDF classes. From [130].

Property name	Comment	Domain	Range
rdf:type	The subject is an instance of a class.	rdfs:Resource	rdfs:Class
rdfs:subClassOf	The subject is a subclass of a class.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	The subject is a subproperty of a property.	rdf:Property	rdf:Property
rdfs:domain	A domain of the subject property.	rdf:Property	rdfs:Class
rdfs:range	A range of the subject property.	rdf:Property	rdfs:Class
rdfs:label	A human-readable name for the subject.	rdfs:Resource	rdfs:Literal
rdfs:comment	A description of the subject resource.	rdfs:Resource	rdfs:Literal
rdfs:member	A member of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:first	The first item in the subject RDF list.	rdf:List	rdfs:Resource
rdf:rest	The rest of the subject RDF list after the first item.	rdf:List	rdf:List
rdfs:seeAlso	Further information about the subject resource.	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	The definition of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:value	Idiomatic property used for structured values.	rdfs:Resource	rdfs:Resource
rdf:subject	The subject of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:predicate	The predicate of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:object	The object of the subject RDF statement.	rdf:Statement	rdfs:Resource

Table D.2: RDF properties. From [130].

resources related by one property are related by another property, then these two properties are themselves hierarchically related by the `rdfs:subPropertyOf` property. On the other hand, if all members of a class are members of another, then these two classes are related by the `rdfs:subClassOf` property. A full list of RDF properties is given in Table D.2.

OWL Web Ontology Language

The OWL Web Ontology Language is a W3C Recommendation designed to provide an ontology language for the Web, that is, a means for explicitly representing terms in vocabularies and the relationship between those terms [291]. While RDF and RDFS allow the representation of *some* ontological knowledge, these are mainly concerned with organization of vocabularies in typed hierarchies [292]. On the other hand, OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and

RDF Schema, through a richer vocabulary that is able to further describe classes, relations between classes, and property types and characteristics [291]. Some examples of OWL constructs are described below.

Classes, class relations and individuals A class defines a group of individuals that belong together because they share some common properties [291]. Such classes are defined in OWL using the `owl:Class` element. If membership in one class automatically implies membership in another, then they are related via the `rdfs:subClassOf` relationship previously defined in RDFS. For example, assume a concept called `ProtocolDataUnit` exists. To define the subset (subclass) of application-layer PDUs (which might be called `Layer7_PDU`, with the use of the prefix "`Layer7`" to denote the application layer of the OSI model), the OWL representation would be as follows:

```
<owl:Class rdf:ID="Layer7_PDU">
    <rdfs:subClassOf>
        <owl:Class rdf:ID="ProtocolDataUnit"/>
    </rdfs:subClassOf>
</owl:Class>
```

The syntax `rdf:ID="Layer7_PDU"` specifies the name of the class, and within the same XML document, the defined class can be referred to as `rdf:Resource="#Layer7_PDU"`.

Unlike in RDF Schema, it is possible in OWL to define classes in terms of Boolean combinations of other classes, through the use of `owl:unionOf`, `owl:complementOf`, or `owl:intersectionOf`. In addition, it is possible to state if classes are disjoint, that is they do not have an intersection (`owl:disjointWith`); or if they are equivalent classes (`owl:equivalentClass`). For example, suppose that a concept called `UnidirectionalFlow`, a subclass of `Flow`, is defined. Additionally (for the purposes of this example), all unidirectional flows either come in pairs (e.g. such as in TCP client-server flow pairs), or are unpaired. The following OWL statements:

```
<owl:Class rdf:about="#UnidirectionalFlow">
    <rdfs:subClassOf rdf:resource="#Flow"/>
    <owl:equivalentClass>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#PairedFlow"/>
                <owl:Class rdf:about="#UnpairedFlow"/>
            </owl:unionOf>
```

```
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```

thus define `UnidirectionalFlow`, a `rdfs:subClassOf Flow`, and further state that it is equivalent to the union of `PairedFlow` and `UnpairedFlow` – that is, *any* unidirectional flow is either a `PairedFlow`, or an `UnpairedFlow`, or both.[1]

OWL also has a predefined class called `owl:Thing`, which encompasses all classes and all individuals in the OWL world, that is, all classes are subclasses of `owl:Thing`, and all individuals are members of `owl:Thing`. The predefined class `owl:Nothing`, on the other hand, is the empty class.

Individuals (members of classes) are introduced by declaring them to be members of a class, using the `rdf:type` property, as in below:

```
<owl:Thing rdf:ID="FedoraCoreLinux"/>

<owl:Thing rdf:about="#FedoraCoreLinux">
    <rdf:type rdf:resource="#OperatingSystem"/>
</owl:Thing>
```

These statements declare that `FedoraCoreLinux` is a member of the class `OperatingSystem`. Equivalently, this may be stated by:

```
<OperatingSystem rdf:ID="FedoraCoreLinux"/>
```

Finally, classes may also be specified by enumerating all its members, using the `owl:oneOf` construct. For example, a class whose members are the different fields of the IPv6 packet header may be defined as:

```
<owl:Class rdf:ID="IPv6PacketHeaderFields">
    <rdfs:subClassOf rdf:resource="#Layer3PDU_HeaderFields"/>
    <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#VersionField"/>
        <owl:Thing rdf:about="#TrafficClassField"/>
```

[1] It would be reasonable to expect that a flow could not be both paired and unpaired at the same time, or in other words, that `PairedFlow` and `UnpairedFlow` are *disjoint*, but this has not asserted to keep the example simple. In addition, any other "bundling" of unidirectional flows has been disregarded.

```
        <owl:Thing rdf:about="#FlowLabelField"/>
        <owl:Thing rdf:about="#PayloadLengthField"/>
        <owl:Thing rdf:about="#NextHeaderField"/>
        <owl:Thing rdf:about="#HopLimitField"/>
        <owl:Thing rdf:about="#SourceAddressField"/>
        <owl:Thing rdf:about="#DestinationAddressField"/>
    </owl:oneOf>
</owl:Class>
```

Properties, property characteristics and property restrictions Properties in OWL define binary relations, either between individuals, or between individuals on one hand and RDF literals or XML Schema datatypes on the other. Properties that relate individuals are called *object properties* while those that relate individuals to datatypes are called *datatype properties*. One may use the `rdfs:domain` and `rdfs:range` constructs from RDFS to specify the domain and range to which a property applies. In addition, hierarchies of properties may also be defined, with one property being defined as as a specialization, or *subproperty* of another, using `rdfs:subPropertyOf` from RDFS.

As an example, assume that a property called `has_IPv4EndpointAddress` describes a relationship between a flow on one hand, and an IPv4 address on the other. In this case the flow could have either originated or terminated at the specified IPv4 address. One could further define a property `has_IPv4DestinationAddress` that specifies that the IPv4 address is the flow's destination address, rather than its source address. As flows and IPv4 addresses related by `has_IPv4DestinationAddress` would also necessarily be related by the property `has_IPv4EndpointAddress`, the former property is thus a subproperty of the latter. This is expressed in OWL by:

```
<owl:ObjectProperty rdf:about="#has_IPv4DestinationAddress">
    <rdfs:subPropertyOf rdf:resource="#has_IPv4EndpointAddress"/>
</owl:ObjectProperty>
```

Properties can have certain characteristics: they may be transitive, symmetric, functional, inverse, or inverse functional. These characteristics are defined in [110] as follows :

- *Transitive property* If a property P is transitive, then for any x, y, and z:

$$P(x, y) \text{ and } P(y, z) \text{ implies } P(x, z)$$

This simply means that if x and y are related by some property P, and y and z are

also related by the same property, then x and z would be related by that property.

- *Symmetric property* If a property P is symmetric, then for any x and y:

$$P(x, y) \text{ iff } P(y, x)$$

- *Functional property* If a property P is functional, then for all x, y, and z:

$$P(x, y) \text{ and } P(x, z) \text{ implies } y = z$$

  Functional properties are also called *single-valued properties* or *features.*

- *Inverse property* If a property P_1 is the inverse of P_2, then for all x and y:

$$P_1(x, y) \text{ iff } P_2(y, x)$$

- *Inverse functional property* If a property P is inverse functional, then for all x, y and z:

$$P(y, x) \text{ and } P(z, x) \text{ implies } y = z$$

This example defines a relationship between flows occurring at different levels of abstraction, for instance, where a flow of IP packets transport a sequence of TCP segments, which in turn transport HTTP messages containing pages marked up in HTML. A property `transports` could be defined as follows:

```
<owl:TransitiveProperty rdf:about="#transports">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <owl:inverseOf rdf:resource="#isTransportedBy"/>
</owl:TransitiveProperty>
```

This object property is defined to be transitive, that is, if a flow of IP packets `transports` a flow of TCP segments, and that flow of TCP segments `transports` an HTTP flow, then this implies that the IP flow `transports` the HTTP flow. While this would seem like a perfectly logical conclusion for humans, the important thing to note is that OWL provides the capability for *machines* to draw such inferences through the use of this explicit property specification, rather than relying on a programmer to embed this logic in program code. Finally, `transports` is defined to be the inverse of the property `isTransportedBy`, which allows one to infer for instance that if a flow of IP packets `transports` a TCP flow, then the TCP flow `isTransportedBy` the IP flow.

In addition to property characteristics, OWL can also express *property restrictions*, which include value constraints and cardinality constraints.

- *Value constraints* include the OWL construct `owl:allValuesFrom` which constrains the relationship of individuals along a certain property, *if* the relationship exists, *exclusively* to members of a given class. For example, in defining a class called called `IPv4_UDP_Flow`[2], which describes flows of IPv4 packets carrying only UDP payloads, the section in the following OWL statements

```
<owl:Class rdf:about="#IPv4_UDP_Flow">
    <rdfs:subClassOf rdf:resource="#IPv4_Flow"/>
    ...
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <owl:InverseFunctionalProperty rdf:about="#hasPDU"/>
            </owl:onProperty>
            <owl:allValuesFrom>
                <owl:Class rdf:about="#IPv4_PDU_UDP_Payload"/>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <owl:InverseFunctionalProperty rdf:about="#hasPDU"/>
            </owl:onProperty>
            <owl:someValuesFrom>
                <owl:Class rdf:about="#IPv4_PDU_UDP_Payload"/>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    ...
</owl:Class>
```

covering the scope of the `owl:allValuesFrom` element stipulate that individuals belonging to this class of flows should only have protocol data units (i.e. IPv4 packets) that contain UDP payloads. The element `rdfs:subClassOf` actually specifies that

[2]The names of the classes and properties used here are not exactly the same as those used in the ontology presented in Section 4.3.2. They have been simplified to improve readability.

the class `IPv4_UDP_Flow` is only a subset of a conceptually "larger" *anonymous class* of individuals that are described by this property restriction [135].

It should be noted that since the `owl:allValuesFrom` restriction does not specify the existence of the relationship, the anonymous class defined by the restriction also includes those individuals that *do not* participate at all in the specified relationship [135]. If this is the case, then the definition for class `IPv4_UDP_Flow` in the previous example is conceptually incorrect, since it would seem to include "flows" that do not have PDUs. To correct this, the `owl:someValuesFrom` property restriction, which specifies the *existence of a* (i.e. at least one) relationship along a given property to an individual that is a member of a given class, is used. If the intersection of the `owl:someValuesFrom` and `owl:allValuesFrom` restrictions is taken, one would obtain an anonymous class that describes individuals satisfying two criteria: they have *at least one* relationship along the specified property, *and* the relationship is exclusively to members of the specified class. This intersection is achieved in the example simply by conjoining the `owl:someValuesFrom` and `owl:allValuesFrom` property restriction statements in OWL.

A third type of value restriction is `owl:hasValue`, which describes a class of individuals for which the property is *semantically equal* in at least one instance to a specified data value or individual. The term "semantically equal" for datatypes means that the lexical representation of the literals maps to the same value, while for individuals it means that they either have the same URI reference or are defined as being the same individual [151].

- *Cardinality constraints* specify the number of elements in a relation. The OWL vocabulary term `owl:maxCardinality` describes a class of all individuals that have *at most* N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint [151]. One the other hand, `owl:minCardinality` describes a class where individuals are related by *at least* N distinct values.

  As an example, to describe a class of SMTP (email) application-layer flows where the number of email recipients are between 30 and 100 (perhaps because the administrator would like to flag them as *possible* spam), the OWL description might contain the following fragment:

```
<owl:Class rdf:about="PossibleSpamFlow">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#has_SMTP_RcptTo" />
```

```
            <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
                100
            </owl:maxCardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#has_SMTP_RcptTo" />
            <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
                30
            </owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

OWL also has a built-in property `owl:cardinality` that describes a class of individuals with relationships along the property concerned with *exactly N* semantically distinct individuals or data values. This construct is however redundant and is included for convenience, as the same effect can be achieved through a pair of pair of matching `owl:minCardinality` and `owl:maxCardinality` constraints with the same value [151].

OWL sublanguages OWL provides three sublanguages, or "species" that have different levels of expressiveness [110].

- *OWL Full* is not really a sublanguage, but the OWL language in its entirety, allowing the use of all of its constructs as well as the unconstrained use of RDF constructs [151]. Furthermore, in OWL Full, `owl:Class` is equivalent to `rdfs:Class`, `owl:Thing` is equivalent to `rdfs:Resource`, and `owl:ObjectProperty` is equivalent to `rdf:Property`. Valid RDF documents may generally be considered to be in OWL Full, unless they comply with the constraints of OWL DL and OWL Lite, as will be described later.

  OWL Full is meant for applications that require both the full vocabulary of OWL and the flexibility of RDF. However, one consequence of this is that OWL Full is *undecidable*, that is, there is no guarantee that a reasoner may derive valid conclusions from statements written in OWL Full in a finite number of steps [151].

- *OWL DL* is a sublanguage of OWL that derives its name from its correspondence with *description logics*, a decidable fragment of first-order logic [110]. It includes all of OWL's language constructs, but with several constraints, including the following [151]:

- a pairwise separation between classes, datatypes, datatype properties, object properties, annotation properties, ontology properties, individuals, data values and the built-in vocabulary;
- no cardinality properties allowed on transitive properties or their inverses or their subproperties;
- well-formed axioms (facts), for instance, all classes and properties referred to within an ontology should be explicitly typed and defined as OWL classes and properties within the ontology, or within an imported ontology); and
- axioms about individual equality and difference should be about named individuals.

These constraints ensure that OWL DL remains *decidable* as previously defined, and also ensures that the language supports the use of reasoners by ontology builders or users [151].

- *OWL Lite* satisfies all the constraints of OWL DL and imposes some additional ones. For example, it excludes the use of `owl:oneOf`, `owl:unionOf`, `owl:complementOf`, `owl:hasValue`, `owl:disjointWith` and `owl:DataRange`. In addition:
  - `owl:equivalentClass` and `rdfs:subClassOf` can only have subjects and objects that are classes or restrictions;
  - `owl:allValuesFrom`, `owl:someValuesFrom` and `rdfs:range` can only have objects that are either class names or datatypes;
  - `rdf:type` can only have objects that are class names or restrictions;
  - only cardinality values of 0 and 1 are allowed; and
  - `rdf:domain` can only have objects that are class names.

  The introduction of additional restrictions in OWL Lite results in a language of lower complexity, supporting the construction of basic subclass hierarchies [151]. This is ideal for the rapid development of ontology tools, and to provide a quick migration path for simple ontologies such as thesauri and other taxonomies [110].