HELSINKI INSTITUTE FOR INFORMATION TECHNOLOGY

STATE OF THE ART IN ENABLERS FOR APPLICATIONS IN FUTURE MOBILE WIRELESS INTERNET

Sasu Tarkoma, Ramya Balu, Jaakko Kangasharju, Miika Komu, Mika Kousa, Tancred Lindholm, Mikko Mäkelä, Marko Saaresto, Kristian Slavov, Kimmo Raatikainen

September, 2004

HIIT Publications 2004-2

STATE OF THE ART IN ENABLERS FOR APPLICATIONS IN FUTURE MOBILE WIRELESS INTERNET

Sasu Tarkoma, Ramya Balu, Jaakko Kangasharju, Miika Komu, Mika Kousa,
Tancred Lindholm, Mikko Mäkelä, Marko Saaresto, Kristian Slavov, Kimmo Raatikainen

Helsinki Institute for Information Technology HIIT
Tammasaarenkatu 3, Helsinki, Finland
P.O. BOX 9800
FI-02015 TKK, Finland
http://www.hiit.fi

HIIT Publications 2004-2
ISBN 951-22-7332-2 (electronic)
ISSN 1458-946X (electronic)
URL: http://www.hiit.fi/publications/pub_files/fc-state-of-the-art-2004.pdf

Contents

List of Figures

Chapter 1

Introduction

One significant trend in software for future mobile systems is the requirement of ever-faster service development and deployment. An immediate implication has been the introduction of various service/application frameworks/platforms. Middleware is a widely used term to denote a set of generic services above the operating system. Although the term is popular, there is no consensus of a definition [ASC+00]. Typical middleware services include directory, trading and brokerage services for discovery, transactions, persistent repositories, and different transparencies such as location transparency and failure transparency. The importance of middleware as a set of generic services above the operating system and transport stack is widely recognized.

The objective of the Fuego Core project is to specify the set of fundamental enabling middleware services for mobile applications on future mobile environments and to implement two research prototypes. The project has adapted a two-level approach to develop the necessary middleware services. On the top level, the *work areas* characterize the long-term vision in research for middleware for the future mobile Internet. On the bottom level, the work areas are further split into *work items* that are addressed in the project.

The work areas in the Fuego Core include

Adaptive Applications
Adaptability is one of the key research areas in nomadic computing. The basic principle of adaptability is simple. When the circumstances change, then the behavior of an application changes according to the desires of the user. Therefore, we need means to collect and present user preferences, which may, in turn, depend on location, time, access device, and properties of connectivity.

The basic principle of adaptability, i.e. the behavior of an application changes when the circumstances change, requires that the system detects changes and notifies about them. Therefore, the generic service elements must include *Environment Monitoring* and *Event Notification*.

In environment monitoring there are three primary issues:

- discovery (which equipment is available),
- service location (which services are available), and
- available capabilities (computing power, various storage capabilities, available capacity on communication paths).

Dynamic Reconfigurable Services

Situations in which a user moves with her end device and uses information services are challenging. Moreover, the nomadic user of tomorrow will not appreciate a static binding between her and an access device; not even in the case of multi-mode access devices that can handle several access technologies including wireless LAN, short-range radio, and packet radio. It must be possible to move a service session (or one endpoint of a service session) from one device to another.

In these situations the partitioning of applications and the placement of different co-operating parts is a research challenge. The support system of a nomadic user must distribute, in an appropriate way, the parts among the end-user system, network elements, and application servers. In addition, when the execution environment changes in an essential and persistent way, it may be beneficial to redistribute the co-operating parts. The redistribution or relocation as such is technically quite straightforward but not trivial. On the contrary, the set of rules on which the detection of essential and persistent changes is based is a challenging research issue.

In the dynamic configuration area we have a huge space of research items. On the conceptual level there are research issues related to profiles, various kinds of context also including the social context, roles, and trust. On the technical level we must solve the problems related to authentication, authorization, and delegation.

Mobile Distributed Information Base

File and information synchronization between different devices is already available but in quite primitive forms. A single information base for a user — possibly different views for her different roles — and for multiple user groups is a fundamental enabler for seamless reconfiguration of the end-user system for a mobile user and for seamless user roaming from one role to another one.

The mobile distributed information base should provide a consistent, efficiently accessible, reliable, and highly available information base. This implies a distributed and replicated world-wide "file system" that also supports intelligent synchronization of data after disconnections. Shared access and support of transactional operations also belong to the list of requirements.

To summarize, the key enablers for mobile distributed information base include

- distributed and replicated world-wide information storage that provides data consistency, efficient and reliable access, and high availability,
- intelligent synchronization after disconnections, and
- distributed mobile transactions with flexible correctness criteria.

Of the fundamental enablers for future mobile applications, the Fuego Core project has focused on seven work items:

Event-based systems

This work item addresses the fundamental principle of adaptability — the behavior of an application changes when the circumstances change. Therefore, the system must detect changes and notify about them. In other words, the middleware solution must provide service elements for Environment Monitoring and Event Notification.

XML issues on profile presentation, protocol, and transport over wireless

XML is starting to be the key presentation format for various kinds of information about capabilities, preferences, and properties. Therefore, middleware for the mobile Internet must provide an efficient way of exchanging XML content and of supporting SOAP.

Intelligent synchronization

Of the fundamental enablers for mobile distributed information base intelligent synchronization is selected as the starting point. The assumption is that an existing storage system, such as Coda, OceanStore, or InterMezzo, can be used in a mobile and wireless environment with file synchronization. The objective is to build mechanisms that take care of decisions on what and when to synchronize. In addition, the use of three-way merging of XML documents is examined in synchronization.

Instant Messaging and Presence

The mobile user should not be required to explicitly define her/his context (presence) information and forward this information to other people. Instead, the computing environment should do this proactively on the user's behalf. But before a proactive presence service becomes a reality for the mobile user, several problems need to be solved.

Mobility and Multi-homing based on Host Identity Protocol (HIP)

Today, there are new requirements for end-host mobility and multihoming, together with the necessity for host-to-host signaling security. Addressing these within the limitations of the current TCP/IP architecture has turned out to be hard; therefore, it may be necessary to do some radical re-engineering of the architecture to bring the TCP/IP protocol suite up to par with the new requirements.

The key elements in considering novel solutions to IP-based end-host mobility and multihoming are a need for a new cryptographic namespace, a need for a new protocol layer between IP and the Internet transport layer (TCP/UDP/SCTP), and privacy and security issues.

Session Initiation Protocol (SIP)

The SIP protocol has been selected as the signaling protocol for third generation wireless networks by 3GPP. The main research items for SIP will be the event packages of SIPPING and SIMPLE. In addition, an analysis of the interworking of the Fuego event system with SIP events will be performed. The investigation will focus on the required event package extensions for SIP, such as filtering and buffering.

Context Modeling

Context awareness can be described as the process of adapting to the current situational characteristics of the computing and user environment. This adaptation is grounded on context modeling, i.e. the process of extracting higher-level context characteristics from the lower-level characteristics, e.g. guessing the type of a social happening based on the time of day, user's calendar information, and the names of the people who are present. One of the central challenges

> in context modeling is due to the fact that the set of possible characteristics of interest is open. Another challenge is that the relations between different characteristics may be temporal in nature, i.e. the relations may vary between situations. We need a way to present and maintain context information in a way that addresses these challenges and is suitable for the mobile environment.

This report examines the relevant related work and background for the work items. Event systems, with an emphasis on mobility, are presented in chapter 2. XML use over wireless is examined in chapter 3 and synchronization in the wireless and mobile environment in chapter 4. The state of the art in mobile presence is presented in chapter 5. The background for Host Identity Protocol is examined in chapter 6. SIP and SIP events are presented in chapter 7, and finally context modeling is investigated in chapter 8.

Chapter 2

Event-based Systems

2.1 Introduction

This chapter presents an overview of event systems and distributed event frameworks with an emphasis on the special requirements presented by mobile computing. By mobile or ubiquitous computing we mean the new field of research created by wireless communication and the introduction of small, mobile devices. Traditionally, event-based systems are based on a number of event sources and event sinks. Sources produce events and they are delivered to event sinks that have a priori registered to receive them. An event-based framework can be decomposed into two essential parts:

- Event detection, which deals with the detection of the occurrence of a particular event of interest.
- Event notification, which is the act of notifying interested parties that an event has occurred.

Distributed architectures are based on middleware that provides the interoperability layer required for heterogeneous cross-operating-system and cross-language operation and communication. Components from different systems and different manufacturers can interoperate using middleware such as CORBA, where interface definitions created using IDL (Interface Definition Language) may be shared.

Many existing platforms employ the synchronous model of method invocation, in which operations are performed on passive objects. This model is insufficient for reactive environments, where components need to react to changes, or events, within the system and give timely responses.

An option would be to poll the states of objects, but polling too frequently burdens the system and polling too infrequently delays the communication [BMH+00]. Asynchronous events support different application types as identified by [BMH+00]:

- Group interaction
- Multimedia support (multimedia control through rules)
- Mobility
- Alarms and exceptions
- Management

Reliable and efficient asynchronous event detection and event notification are vital for the development of the next-generation distributed software for mobile Internet-aware devices. Event frameworks provide a plug-and-play architecture for creating distributed applications.

Currently middleware solutions, such as Java, from the desktop world are being introduced into the wireless world, where the requirements are different. Small and wireless devices have limited capabilities compared to desktop systems: their memory, performance, battery life, and connectivity are limited and constrained. The requirements of mobile computing need to be taken into account when designing an event framework that integrates with mobile devices. From the mobility and wireless viewpoint event systems can be divided into three distinct categories:

1. Traditional event systems designed for fixed network operation
2. Event systems that support intermittent clients using a client-server protocol and possibly roaming between access nodes
3. Ad-hoc networks

The first category is the most researched and most of the architectures presented in this chapter fall into this category. Several architectures support intermittent clients and roaming between access nodes. Ad-hoc event architectures are currently emerging, and they are only mentioned briefly in this chapter.

From the small device point of view, message queuing is a frequently used communication method because it supports disconnected operation. When a client is disconnected, messages are inserted into a queue, and when a client reconnects the messages are sent. The distinction between popular message-queue-based middleware and notification systems is that message-queue-based approaches are a form of directed communication, where the producers explicitly define the recipients. The recipients may be defined by the queue name or a channel name, and the messages are inserted into a named queue, from which the recipient extracts messages.

Notification-based systems extend this model by adding an entity, the event service or event dispatcher, that brokers notifications between producers of information and subscribers of information. This undirected communication supported by the notification model is based on message passing and retains the benefits of message queuing. In undirected communication the publisher does not necessarily know which parties receive the notification. This also applies to message-oriented middleware that supports publish-subscribe-style communication [SAS01], such as the Java Message Service (JMS) [Sun01].

Undirected communication decouples producers and consumers from each other. In addition, many systems support filtering and pattern detection that are used to reduce the amount of transmitted information and to improve the accuracy of notifications. Content-based routing is flexible because it does not require configuration information pertaining to channel names. Undirected communication may also be used to deliver the same set of information to a number of client devices. However, this requires associating user subscription information with a set of devices [SAS01, CDN01].

This chapter is structured as follows: section 2.2 introduces event models, event routing, and a number of requirements for mobile clients, section 2.3 presents event standards and specifications such as the CORBA Notification Service, JMS, and the COM+ and .NET event models, section 2.4 presents research prototypes and examines the support for disconnected operation and mobility in each of the presented event systems, and finally, section 2.5 presents the conclusions.

2.2 Event Models

Event models consist of event sources, event listeners, notification services, filtering services, and event storage and buffering services. In addition, there may be one or more authentication schemes to enforce security and access control. This section focuses on the general definition of events and event models.

2.2.1 Events

An event represents any discrete state transition that has occurred and is signalled from one entity to a number of other entities. For example, a successful login to a service or the firing of detection or monitoring hardware can be seen as events.

The firing of each event is either deterministic or probabilistic. A source can generate a signal every second making it deterministic. A stochastic source follows some probabilistic model that can be described using, for example, a Markov chain. Both event qualities can be modeled by building statistical or stochastic models of the firing behavior of the event source. For example, a correlation analysis can be made between a series of event occurrences in time or between two event sources. Such an analysis would measure how strongly one event implies the other or how two event source firings are related.

Events may be categorized by their attributes, such as which physical property they are related to. For instance spatial events and temporal events denote physical activity. Moreover, an event may be a combination of these, for example an event that contains both temporal and spatial information.

Events can be categorized into taxonomies on their type and complexity. More complex events, called compound events, can be built out of more specific simple events. Compound events are important in many applications. For example, a compound event may be fired

- in a hospital, when the reading of a sensor attached to a patient exceeds a given threshold and a new drug has been administered in a given time interval,
- in a location tracking service, where a set of users are in the same room or near the same location at the same time, or
- in an office building, where a motion detector fires and there has been a certain interval of time after the last security round.

Event-based interaction can be

- discrete or
- continuous, as event streams.

Events can also have different prioritizations. Event aging assigns an expiry time to each event notification. Event expiration prevents the spreading of obsolete information.

2.2.2 Event Model

The standard models for client/server communication in distributed object computing are based on synchronous method invocations. For example, COM+, Java RMI, and CORBA use synchronous calls (CORBA 3.0 supports asynchronous invocations). This approach has several limitations [GCSO01]:

- Tight coupling of client and server lifetimes. The server must be available to process a request. If a request fails the client receives an exception.
- Synchronous communication. A client must wait until the server finishes processing and returns the results. The client must be connected for the duration of the invocation.
- Point-to-point communication. Invocation is typically targeted at a single object on a particular server.

Mobile clients and large distributed systems motivate the use of asynchronous and anonymous one-to-many models of communication. Models based on events address the limitations of the standard client/server paradigm by introducing two roles: consumers and producers. Since event models employ differing technical terms, in this chapter we consider event consumers, listeners, sinks, and respectively event producers, sources, and suppliers to be synonymous.

The event model consists of event listeners and event sources. A listener expresses interest in an event supported by an event source and registers to receive notifications of that event based on a set of parameters. Figure 2.1 presents a general model of the listener-source paradigm, where the actual filtering and notification are treated as a black box, which can reside either on the source or on the network. Ideally, the event source does not have knowledge of all the parties that are interested in a particular event.

The event system is a logically centralized component that may be a single server or a number of federated servers. In a distributed system consisting of many servers, there are two approaches for connecting sources and listeners:

- The event service supports subscription of events, and it routes registration messages to appropriate servers (for example, using a minimum spanning tree). One optimization to this approach is to use advertisements, messages that indicate the intention of an event source to offer a certain type of event, to optimize event routing.

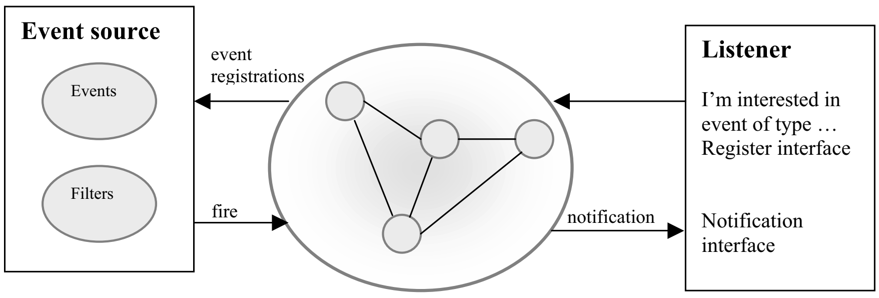

Figure 2.1: General model of the event source and event listener. Event source fires events, and the listener is notified using some mechanism on the network or in the client.

- Some other means of binding the components is used, for example, a lookup service.

In this context, by event listener we mean an external entity that is located on a physically different node on the network. However, events are also a powerful method to enable inter-thread and local communication, and there may be a number of local event listeners that wait for local events.

2.2.3 Routing

Event routing requires that store-and-forward type of event communication is supported within the network on the access nodes (or servers). This calls for intermediate components called event routers. Each event source is connected to at least one router. Each router needs to know a suitable subset of other routers in the domain.

In this approach the request, in the worst case, is introduced at every router to get a full coverage of all message listeners. This is not scalable, and the routing needs to be constrained by locality or by hop count. Effective strategies to limit event propagation are zones used in the ECO architecture [HMN$^+$00], the tree topology used in JEDI, or the four server configurations addressed in the Siena architecture. Siena broadcasts advertisements throughout the event system, subscriptions are routed using the reverse-path of advertisements, and notifications are routed on the reverse-path of subscriptions. IP Multicast is also a frequently used network-level technology for disseminating information and works well in closed networks, however, in large public networks multicast or broadcast may not be practical. In these environments universally adopted standards such as TCP/IP and HTTP may be better choices for all communication [IBM02a].

2.2.4 Content-based Routing

Events are published in a named channel, or in an infrastructure of one or more routers that can use the content of the events in making the forwarding decision. Named channels are also called topics, and they represent an abstraction of numeric network addressing mechanisms. With content-based addressing clients can change their interests without changing the addressing scheme. With channel-based messaging, new channels need to be added to the address space.

Content-based The routing decision is made based on the content, for example strongly typed fields in the event message.

Subject-based The routing decision is made based on the subject of the event.

Channel-based (or topic-based) The routing decision is made based on the channel on which the event is published. A channel is a discrete communication line with a name.

The producers and consumers must agree on a channel. Content-based and subject-based are more flexible than channel-based messaging, because this agreement is not necessary. Channel-based messaging, however, allows the use of IP multicast groups. The subjects can be allocated to multicast addresses. Channel-based routing can be emulated with content-based systems by limiting to a universally defined subject field.

Content-based event routing has been proposed as one of the requirements for advanced applications, in particular for mobile users [CW01]. Content-based routing takes place above the network level (level 3) and can be based on e.g. IP multicast networks. In the content information model, the users subscribe to information based on their preferences. The information, when it is available, is then delivered based on these preferences. The subscription paradigm abstracts the publishers of information from the receivers: information is not published to a set of addresses.

Work has been done in using multicast networks to deliver the information to the subscribers [CW01] using multicast addresses. The granularity and flexibility of this approach depends on the size and number of the virtual multicast addresses. As an alternative Carzaniga and Wolf present an application-level information broker with a rich information selection capability. They define a content-based addressing scheme by considering the predicates that define subscriptions as the destination addresses. Datagrams are implicitly addressed to a node by their content. The predicate model is a set of boolean functions imposed on the datagram model. Content-based routing is done using an algorithm that uses a forwarding table, which is a map of interfaces to their receiver predicates.

Content-based systems are contrasted with channel-based and subject-based systems, because the selection is done based on the whole content. The other strategies offer only a set of well-defined attributes for selection purposes.

Filtering

Filtering reduces the number of events sent from the sources to the listeners by matching events against a template. Those events that match the template are forwarded to the listeners. Matching is usually done on single events, but may be also performed on compound events. Filtering improves the scalability of the system. Also, the location of the filtering of events affects the scalability of the framework. Here we face two separate issues: the filtering of simple events and the filtering of compound events. Both kinds of event filtering can be done at several locations:

- At a centralized server (client-server)
- At the listener
- At the event source
- In the infrastructure (event routers)

Source-side filtering is more scalable than a centralized server or filtering at the listener. Schemes that use multicasting and listener-side filtering place the burden on listeners and the communication infrastructure.

Quality of Service

Applications based on event-style communication have varying reliability requirements. The event system may support semantics ranging from "at-most-once" to "exactly-once". In addition, there may be availability, performance, scalability, and throughput requirements. The diverse nature of requirements calls for a number of implementations optimized for different sets of requirements.

Taxonomy

Event models can be grouped into a taxonomy by their properties. As contrasted with the client-server paradigm, event models involve one-to-many communication. Other important aspects for event model classification are [Mei00]:

- Does the model support distributed operation, local operation, or both? In a centralized event model the event sources and listeners are located on the same host, whereas in the distributed model they can be located on different hosts.
- Support for detecting composite events (compound events). Compound events require more complicated filtering and history mechanisms.
- Support for Quality of Service requirements, for example, delivery semantics (best-effort, at-most-once, . . .).
- Support for typed events, generic events, or both. Typed events have a well-defined structure, for example a set of ordered strings, and generic events do not have an expressive structure (datatype any).
- How decoupled the event listeners are from the event sources?
- Is the model subscription-based or advertisement-based?
- Support for channel-based, subject-based, or content-based routing.

Additional aspects are:

- Support for wireless systems and disconnected operation.
- Does the model support event routing, direct notification, etc.?
- How are interests defined and discovered? Not all models include discovery functionality.

Figure 2.2 and Figure 2.3 present an example taxonomy based on the event architectures explored in section 2.4.

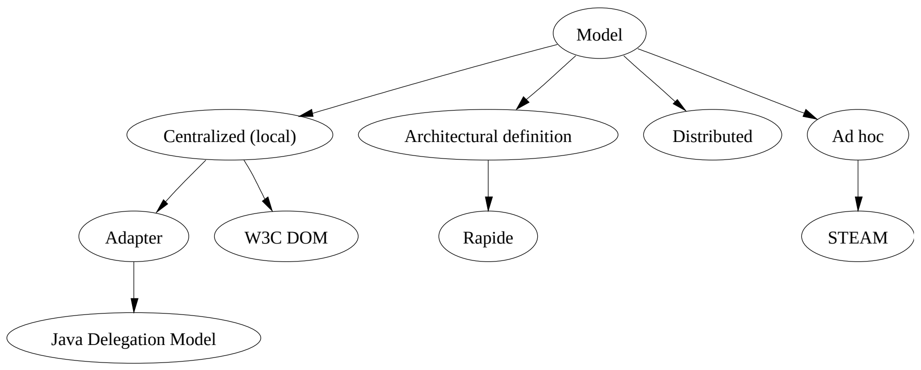

Figure 2.2: Example event model taxonomy.

2.2.5 Requirements for Mobile Computing

The mobile environment poses several challenges for detecting and distributing events:

- The network connections are intermittent.
- Bandwidth may vary greatly depending on the connection. This effectively puts constraints on the number of events that can be sent during a certain time interval, how timely the events are, and how reliably they can be communicated to the other party.
- The devices may have limited system resources (CPU, memory, storage) and may not have capability to pre-process events but send them as they occur. This motivates an event service located on the fixed network that provides high-level event support for mobile clients.
- The mobile clients may move to a different geographic location or roam in a different network. It is preferable that the event service works after a change in connectivity or service domain.
- The user may wish to share a set of subscriptions between different devices.

We need to consider the following requirements:

- Timely delivery of events, timely being defined in a suitable, application-specific context.

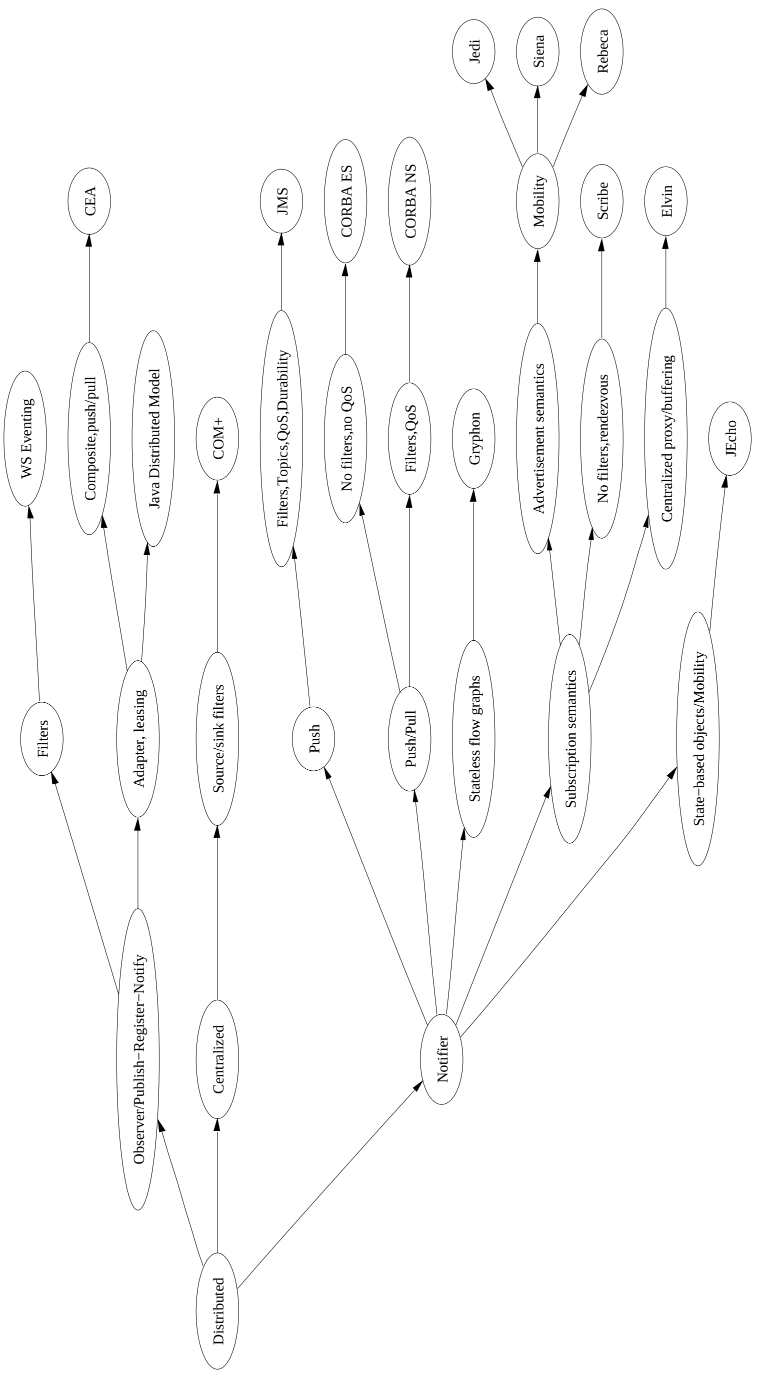

Figure 2.3: Distributed event systems in the taxonomy.

- Reliable delivery of events. Events must be delivered as they are published. Events may not become lost.
- Events need to be processed asynchronously.
- Events need to be monitored and notified across domains.
- Event ordering must be preserved: if events are used for synchronizing applications either a causal or a total order is required.

In order to support reliable and fault-tolerant event notification, the event sources need to provide reliable persistent storage and buffer events. This is more realizable with fixed network servers, because the mobile clients do not necessarily have persistent storage.

From the device point of view:

- Transmission cost depends on the number of bytes transmitted and the battery life (transmission requires energy).
- How much event history is stored within the device? A distributed event service should be used only for external purposes, not for internal monitoring.
- How to deliver notifications to the device in different networks and protocols? For instance, the bearer may not support push-type communication.

A mediator [BMH+00] (a proxy) can prevent the disconnected mobile user from missing events. In this case, the mediator registers events on behalf of the mobile client and buffers the event notifications. The size of the accumulated set of events may be fairly large. Therefore, the client needs some way to prune the event history and decide what events are crucial for delivery. On the other hand, the client can decide what events are registered, and may deregister unimportant events when the bandwidth is low or costly.

The events may also have a limited temporal existence according to user, system, or application requirements. Time-to-live (TTL) timers and hop counters can be used to remove obsolete events.

From the security point of view we have to take into account

- Encrypting subscriptions and notifications.
- Securing communications between various communicating entities.
- Access control and authentication.

2.3 Event Standards and Specifications

This section presents event standards and specifications. We start from the standard centralized event model in Java, and continue with the Distributed Event Model in Java. We present the Java Message Service in subsection 2.3.3, subsection 2.3.4 presents the CORBA Event Service and subsection 2.3.5 the Notification Service. In subsection 2.3.6 we examine the CORBA Management of Event Domains. We also examine W3C DOM events, Web Services events, COM+ and .NET, and the WebSphere MQ architecture.

2.3.1 Java Delegation Event Model

The Java Delegation Event Model was introduced in the Java 1.1 Abstract Windowing Toolkit (AWT) and serves as the standard event processing method in Java. The model is also used in the Java Beans architecture and supported in the PersonalJava and EmbeddedJava environments.

In essence, the model is centralized and a listener can register with an event source to receive events. An event source is typically a GUI element and fires events of certain types, which are propagated to the listeners. Event delivery is synchronous, so the event source actually executes code in the listener's event handler. No guarantees are made on the delivery order of the events [Mei00].

The event source and event listener are not anonymous, however, the model provides an abstraction called an adapter, which acts as a mediator between these two actors. The adapter decouples the source from the listener and supports the definition of additional behavior in event processing. The adapter may implement filters, queuing, and QoS controlling.

2.3.2 Java Distributed Event Model

The Distributed Event Model of Java is based on Java Remote Method Invocation (RMI) that enables the invocation of methods in remote objects. This model is used in Sun's Jini architecture. The architecture of the Distributed Event Model is similar to the architecture of the Delegation Model with some differences.

The model is based on the Remote Event Listener, which is an event consumer that registers to receive certain types of events in other objects. The specification provides an example of an interest registration interface, but does not specify such. The Remote Event is the event object that is returned from an event source (generator) to a remote listener. Remote events contain information about the occurred event, a reference to the event generator, a handback object that was supplied by the listener, and a unique sequence number to distinguish the event globally. The model supports temporal event registrations with the notion of a lease (Distributed Leasing Specification). The event generators inform the listeners by calling the listeners' notify method. The specification supports Distributed Event Adaptors that may be used to implement various QoS policies and filtering.

The handback object is the only attribute of the Remote Event that may grow to unbounded size. It is a serialized object that the caller provides to the event source; the programmer may set the field to null. Since the handback object carries both state and behavior it can be used in many ways, for example to implement an event filter at a more powerful host than the event source. A mediator component can register to receive events and give a filter object to the source. Upon event notification, the filter is handed back and the mediator can use it to filter the event before handing it to the original event listener.

The specification supports recovery from listener failures by the notion of leasing. A lease imposes a timeout for event registrations. This is used to ease the implementation of distributed garbage collection. Since this model relies on RMI, it is synchronous. Each notification contains a sequence number that is guaranteed to be strictly increasing.

2.3.3 Java Message Service

Java Message Service (JMS) [Sun01] defines a generic and standard API for the implementation of message-oriented middleware. The JMS API is an integral part of the Java Enterprise Edition (J2EE) version 1.3. The J2EE supports the message-driven bean, a new kind of bean that enables the consumption of messages. However, JMS is an interface and the specification does not provide any concrete implementation of a messaging engine. The fact that JMS does not define the messaging engine or the message transport gives rise to many possible implementations and ways to configure JMS. JMS supports a point-to-point (queues) model and a publisher/subscriber (topics) model. In the point-to-point model only one receiver is selected to receive a message, and in the publisher/subscriber model many can receive the same message.

The JMS API can ensure that a message is delivered only once. At lower levels of reliability an application may miss messages or receive duplicate messages. A standalone JMS provider (implementation) has to support either point-to-point or the publish/subscribe approach, or both. Normally, JMS queues and topics are maintained and created by the administration rather than application programs. Therefore the destinations are seen as long-lasting. The JMS API also allows the creation of temporary destinations that last only for the duration of the connection.

The point-to-point communication model consists of receivers, senders, and message queues. Each message queue is addressed to a particular queue, and receivers extract messages from the queues. Each message has only one consumer and the client acknowledges the successful delivery of a message to the component that manages the queue. In this model there are no timing dependencies between a sender and a receiver; it is enough that the queue exists.

In addition, the JMS API allows the grouping of outgoing messages and incoming messages and their acknowledgements to transactions. If a transaction fails, it can be rolled back.

In the publish/subscribe model the clients address messages to a topic. Publishers and subscribers are anonymous, and messaging is usually one-to-many. This model has a timing dependency between consumers and producers. Consumers receive messages after their subscription has been processed. Moreover, the consumer must be active in order to receive messages. The JMS API provides an improvement on this timing dependency by allowing clients to create durable subscriptions. Durable subscriptions introduce the buffering capability of the point-to-point model to the publish/subscribe model. Durable subscriptions can accept messages sent to clients that are not active at the time. A durable subscription can have only one active subscriber at a time.

Messages are delivered to clients either synchronously or asynchronously. Synchronous messages are delivered using the receive method, which blocks until a message arrives or a timeout occurs. In order to receive asynchronous messages, the client creates a message listener, which is similar to an event listener. When a message arrives the JMS provider calls the listener's onMessage method to deliver the message.

JMS clients use JNDI (Java Naming and Directory Service) to look up configured JMS objects. JMS administrators configure these components using facilities specific to a provider. There are two types of administered objects in JMS: ConnectionFactories, which are used by clients to connect with a provider, and Destinations, which are used by clients to specify the destination of messages.

JMS messages consist of a header with a set of header fields, properties that are optional header fields (application-specific, standard properties, provider-specific properties), and a body that can be of several types.

Message selection is supported by filtering the message header against the given criteria using an SQL grammar. A JMS message selector allows clients to define the messages they are interested in. Headers and properties need to match the client specification in order to be delivered to that client. Message selectors cannot reference values embedded in the message body. An example is “JMSType=’stock’ AND company=’abc’ AND stockvalue > 100”.

JMS supports five different messages types: Map, Object, Stream, Text, and Bytes. MapMessage is a set of name/value pairs, where names are strings and values are primitive Java types. ObjectMessage is a message containing a serializable Java object. StreamMessage is a stream of sequential Java primitive values. TextMessage represents an instance using the java.lang.String class and can be used to send and receive XML messages. BytesMessage is a stream of bytes.

Typically a JMS client creates a Connection, one or more Sessions, and a number of MessageConsumers and MessageProducers. Connections are created in the stopped mode. After a connection is started (start() method) messages start arriving to the consumers associated with that connection. A MessageProducer can send messages while a Connection is stopped. A Session is a single-threaded context for consuming and producing messages. Sessions act as factories for creating MessageProducers, MessageConsumers, and temporary destinations. JMS defines that messages sent by a session to a destination must be received in the order in which they were sent.

Messages are acknowledged automatically in the transactional mode (supported by the Java Transaction API), however, if a session is not transacted there are three possible options for acknowledgement: lazy acknowledgment that tolerates duplicate messages, automatic acknowledgement, and client-side acknowledgement. In persistent mode delivery is once-and-only-once, and in non-persistent mode the semantics are at-most-once.

JMS messaging proceeds in the following fashion:

1. Client obtains a Connection from a ConnectionFactory

2. Client uses the Connection to create a Session object

3. The Session is used to create MessageProducer and MessageConsumer objects, which are based on Destinations.

4. MessageProducers are used to produce messages that are delivered to destinations.

5. MessageConsumers are used to either poll or asynchronously consume (using MessageListeners) messages from producers.

The JMS API (1.0.2b) does not address load balancing, fault tolerance, error notification, administration, or security. JMS implementations are available from many vendors, such as IBM (MQSeries), Sun Microsystems (J2EE), The ExoLab Group (OpenJMS), SoftWired (iBus//Mobile), and Oracle (8i and later).

The latest JMS version is 1.1, which incorporates changes approved by a Java Community Process program Maintenance Review that closed on March 18, 2002. In JMS 1.0.2 client code must use the queue and topic interfaces, and it is impossible to reuse queue clients with topics. JMS 1.1 supports client code that works simultaneously with either the point-to-point or publish/subscribe domains. Queues and topics can be accessed through the same session and thus in the same transaction.

JMS and CORBA Interoperability

The communication models of JMS and CORBA are similar, however, integration is necessary in the areas of message conversion, filtering, and the incorporation of point-to-point mode, which uses queues (CORBA uses publish-subscribe). The Notification Service supports structured events defined in IDL, and JMS supports the five different message formats.

OMG is working on a Notification Service / JMS Interworking document [OMG02]. The Request For Proposals (RFP) dealt with mappings between message types, reconciliation between different QoS properties, the ability to maintain transactional message contexts across the services, and implementations which facilitate end-to-end messaging between the services. The submission document has been replaced with an OMG Final Adopted Specification, which is currently in the finalization phase.

The specification defines a bridge that manages and interconnects an event channel with a JMS destination. The principles behind the Bridge IDL definitions were to provide backward compatibility with the programming models of NS and JMS. The Bridge is a stateful entity that mediates messages between the two systems. Structured events are used to improve performance. The Bridge is also used to automate the connection setups between channels and destinations. A BridgeFactory object supplies Bridge objects depending on the parameters: channel, destination, type of communication (push/pull), and message type (sequence, single). Since JMS does not support pull at the source side, this is not supported.

In the implementation of PrismTech's OpenFusion [Pri01], the JMS event producer is extended by a client-side library that transforms JMS messages to CORBA Notification Service structure events. JMS consumers may use push and pull, but the consumers of the Notification Service may only use one of these two approaches.

JMS only allows clients to specify filters on the message properties. To keep the information filterable, this data needs to be included in the filterable body of a structured event. The JMS message interface supports three attributes that are also supported in the Notification Service:

1. DeliveryMode (persistent, non-persistent which maps to best effort in CORBA NS)

2. Expiration (expiration in milliseconds, set to QoS in the variable Timeout)

3. Priority (Mapped to notification Priority QoS in the variable header)

Other user-defined name-value pairs are converted to IDL using the standard primitive mapping.

Since Notification Service uses the Extended Trader Constraint Language and JMS uses the where clause of SQL92, the Notification Service needs to be extended to support SQL92.

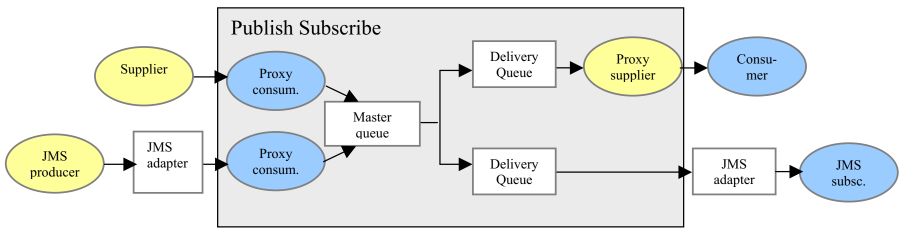

Figure 2.4: The OpenFusion Notification Service with JMS publish-subscribe interoperability.

Wireless JMS

The iBus//Mobile software from SoftWired consists of a server-side gateway for mobile clients and a JMS compatible messaging server (iBus//-MessageServer). The gateway enables communication between a wide variety of devices running different operating systems, such as PalmOS,

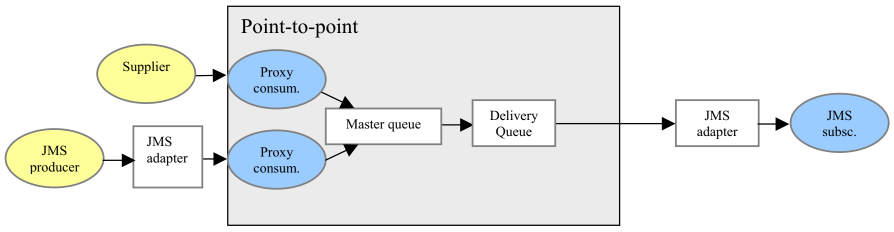

Figure 2.5: The OpenFusion Notification Service with JMS point-to-point interoperability [Pri01]

Symbian, and PocketPC. The gateway supports communication over SMS, WAP, TCP, UDP, and GPRS. The system supports corresponding Java virtual machines, J2ME (CLDC and CDC), PersonalJava, and J2SE [R+01a].

All communication between the clients and the gateway is transmitted in binary form. From the JMS provider's viewpoint the gateway is a regular JMS client and from the client's viewpoint the gateway is a communication hub and a wrapper for different transport and representation formats. In the case of SMS the gateway accepts the incoming messages and a component within the service domain can respond with SMS.

The client side library takes a minimum of 70k and at runtime the CLDC version takes a minimum of 50k of Java heap (as a comparison, a 8MB Palm has a 150k Java heap). The iBus system supports security in the form of access control, certificates, and symmetric/asymmetric keys. Cryptographic functions are supported through third-party libraries. If the bearer does not support push-type connections, one connection is used for sending client data to the server and another connection is used for communication from the gateway to the client. Each HTTP request goes over the first connection: send data to the servlet, and return. The second connection is open and blocks until there is traffic; after receiving messages the connection is immediately re-established. The underlying library hides the differences between the protocols.

2.3.4 CORBA Event Service

The CORBA Event Service specification (current version 1.1) defines a communication model that allows an object to accept registrations and send events to a number of receiver objects [Sie99]. The Event Service supplements the standard CORBA client-server communication model and is part of the CORBAServices that provide system level services for object-based systems. In the client-server model illustrated in Figure 2.6, the client makes a synchronous IDL operation on a specified object at the server. The event communication is unidirectional (using CORBA one-way operations) [OMG01a].

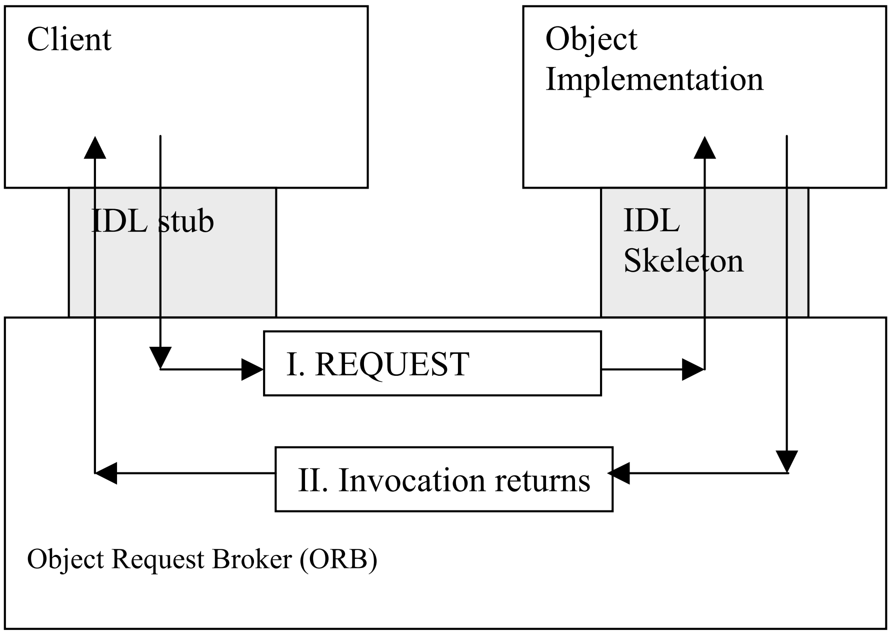

Figure 2.6: The standard CORBA client-server model of invoking operations from client to the target object.

The Event Service extends the basic call model by providing support for a communication model where client applications can send messages to arbitrary objects in other applications. The Event Service addresses the limitations of synchronous and asynchronous invocation in CORBA.

The specification defines the concept of events in CORBA: an event is created by the event supplier and is transferred to all relevant event consumers. The set of suppliers is decoupled from the set of consumers, and the supplier has no knowledge of the number or identity of the consumers. The consumers have no knowledge of which supplier generated the event.

The Event Service defines a new element, the event channel, which asynchronously transfers events between suppliers and consumers. Suppliers and consumers connect to the event channel using the interfaces supported by the channel. An event is a successful completion of a sequence of operation calls made on consumers, suppliers, and the event channel.

The event channel performs the following functions:

- It allows consumers to register interest in events and stores the registration information.
- It accepts events generated by suppliers.
- It forwards events from suppliers to registered consumers.

The Event Service is defined to operate above the ORB architecture: the suppliers, the consumers, and the event channel may be implemented as ORB applications and events are defined using standard IDL invocations.

Push and Pull

The CORBA Event Service provides two models for initiating the transfer of events between suppliers and consumers. The first model is the push model, in which suppliers send events to consumers (Figure 2.7). In this case the suppliers are active and the consumers passive. Moreover, the event channel actively delivers events to the consumers. In the second model, the pull model (Figure 2.8), the consumers request events from the suppliers. A supplier actively waits for pull requests to arrive. Upon the arrival of a pull request, the requested event is generated and sent to the pulling consumer. CORBA supports both blocking and non-blocking pull.

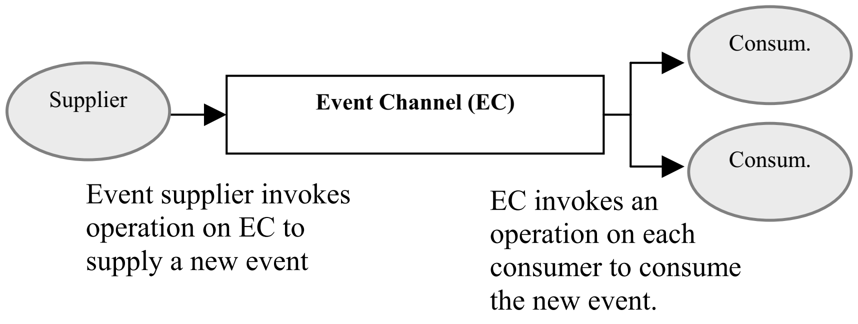
Supplier
Event Channel (EC)
Consum.
Consum.
Event supplier invokes operation on EC to supply a new event
EC invokes an operation on each consumer to consume the new event.

Figure 2.7: Example of an event propagation implementation.

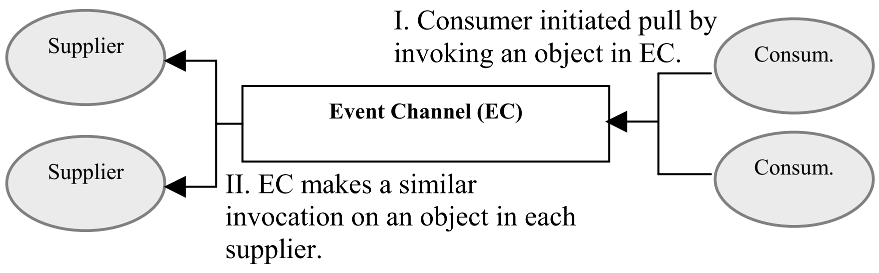
Supplier
Supplier
I. Consumer initiated pull by invoking an object in EC.
Event Channel (EC)
II. EC makes a similar invocation on an object in each supplier.
Consum.
Consum.

Figure 2.8: Pull Model and the Event Channel.

The Hybrid Model

It is also possible to mix the push and pull models in one application, because the event channel decouples the consumers and suppliers from each other. It is possible to connect suppliers using the push model and consumers using the pull model. In the hybrid model, the event channel does not take an active role in delivering the event to the consumers.

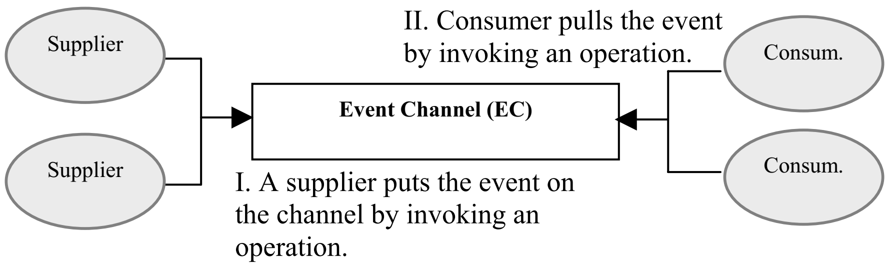
Supplier
Supplier
II. Consumer pulls the event by invoking an operation.
Event Channel (EC)
I. A supplier puts the event on the channel by invoking an operation.
Consum.
Consum.

Figure 2.9: The hybrid model mixing Push and Pull models.

Connecting Suppliers and Consumers

The Event Service specification does not include a mechanism for locating or discovering consumers or suppliers, however, it provides the administrative operations for connecting the suppliers and consumers. Each new event consumer added to the event channel returns a proxy supplier. The proxy supplier follows the supplier interface and has a method for connecting a consumer to the proxy supplier. Each new event supplier added to the event channel returns a proxy consumer. The proxy consumer has a method for connecting to the proxy supplier.

A supplier is registered by taking a proxy consumer from the event channel and connecting it with the supplier. Similarly, an event-receiving application takes a proxy supplier from the event channel and connects to it by providing a consumer. Each admin object is a factory that creates the proxy interface that is used in connecting the clients and the event sources. Consumer admins create proxy suppliers and supplier admins create proxy consumers.

Typed and Untyped Event Communication

The data of an event can be passed as invocation parameters or return values. Events are not objects, because the CORBA object model does not support passing objects by value (CORBA 2.3 supports valuetypes). Event data is application-specific and can be either untyped or typed.

In untyped communication the event is propagated by invoking a series of generic push and pull operations. The push operation takes a single parameter of type any, which allows any IDL defined datatype to be propagated, and stores the event data. The pull operation has no parameters and transfers event data in its return value, which is of type any. In untyped communication both the supplier and the consumer applications need to agree on the data format of the event.

In typed event communication events are propagated through an application-specific interface created by the programmer in IDL. The programmer defines the interface for event propagation that is used by consumers and suppliers. Parameters can be of any suitable datatype supported by the IDL language.

To setup typed push-style communication, the consumers and suppliers exchange object references (TypedPushConsumer and PushSupplier). The supplier invokes a method to get a reference that supports the typed consumer interface. The particular reference is associated with the TypedPushConsumer interface and needs to be agreed on by both the consumer and the supplier. The supplier uses this reference to invoke operations on the consumer.

In the typed pull model consumers request event information using some mutually agreed interface. The parties exchange the PullConsumer and TypedPullSupplier interfaces, and an object reference supporting the typed interface is obtained. Once the reference is obtained, the consumer can invoke operations on the supplier.

Discussion

The CORBA Event Service supports different implementations of the Event Channel, and this allows a wide range of approaches for implementing Quality of Service and delivery issues.

The CORBA Event Service addresses some of the problems of the standard CORBA synchronous method invocations by decoupling the interfaces and providing a mediator for asynchronous communication between consumers and suppliers. The supplier does not have to wait for the event to be delivered to the consumer. Moreover, the event channel hides the number and identity of the consumers from suppliers using the proxy objects (transparent group communication). The supplier sends events to its proxy consumer, and the consumer receives events from its proxy supplier. The event consumer and supplier interfaces support disconnection.

The specification does not address several important issues, such as Quality of Service support. Applications may have requirements for event notification in terms of reliability, ordering, priority, and timeliness. Furthermore, the specification does not provide a system for event filtering. Event filtering needs to be implemented using a proprietary system within the event channel by adding a mechanism for selective event delivery. Event channels can be composed, because they use the same consumer/supplier interfaces. An event channel can push an event to another event channel. Typed event channels can be used to filter events based on event type [Bar01, OMG01a].

In addition, the specification does not address compound events, but suggests that complex events may be handled by creating a notification tree and checking event predicates at each node of the tree. The drawback of the tree is that the number of hops needed to deliver an event increases.

The use of proprietary event service implementations restricts the interoperability of applications. Applications that use one proprietary event service implementation may not interoperate with another application that is based on a different event service implementation.

2.3.5 CORBA Notification Service

The CORBA Notification Service (current version 1.0.1) [OMG01c] extends the functionality and interfaces of the Event Service to support better interoperability [Bar01]. One of the most significant additions to the Notification Service is event filtering. Filters allow consumers to receive particular events that match certain constraint expressions. Filtering reduces the number of events sent to the consumers and improves the scalability of the event handling system.

Figure 2.10 presents the components of the CORBA Notification Service, which derive from the Event Service discussed in the previous section. The event channel has been extended to support a number of admin objects. The Notification Service allows the definition of filters at the proxies. Moreover, each admin object is seen as the manager of the set of proxies it has created. Admin objects may be associated with QoS properties and filter objects. The QoS properties and filter objects of the admin object are transferred to each proxy it creates, however, the QoS properties may be changed on a per-proxy basis.

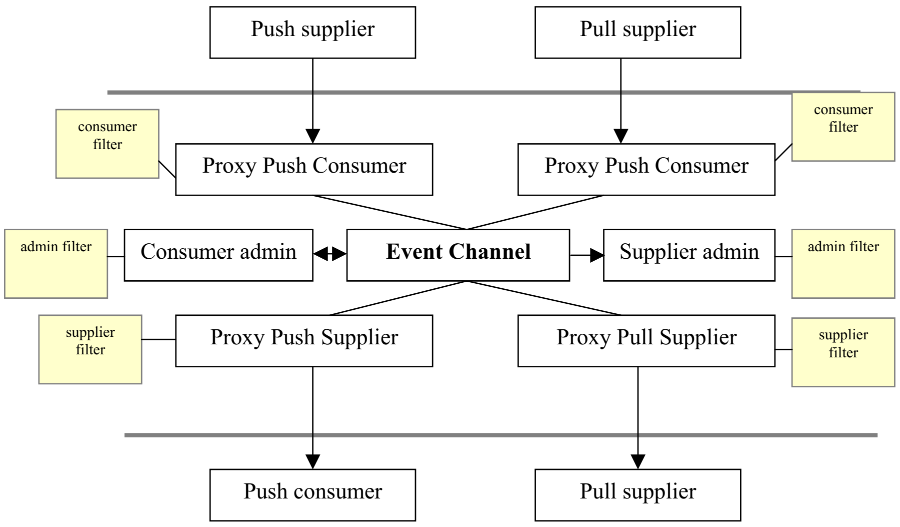

Figure 2.10: Components in the CORBA Notification Service [GCSO01].

Filters

Filters are CORBA objects that support the addition, modification, and removal of constraints. Constraints are used to match event message values and refer to variables that are part of the event notification message. Constraints are either event types or written in a constraint language. Variable names can refer to all parts of the current notification. The current notification is expressed with the dollar sign '$'.

A sample notification constraint:

```
$.type_name == StockAlert
$.market_name == 'NASDAQ'
$.ticker == 'Company'
$.price > '100' or $.price < 80
```

The default constraint grammar is Extended TCL (Trader Constraint Language specified by the Trading Service). The Event Notification specification adds the notion of mapping filter objects. Each proxy supplier may have an association with a mapping filter object, which affects the priority and the lifetime property of the events it receives.

Quality of Service (QoS)

The Notification Service defines standard interfaces that allow the control of characteristics over the delivery of the notification. Service characteristics at different levels in the protocol stack are represented using name/-value pairs. QoS properties, tuples of the form <String, Any>, can be used with an event channel, admin objects, proxy suppliers, proxy consumers, and message instances.

Characteristics include:

- Discard policy that determines which notifications are discarded when resource limits apply (queues are full).
- Earliest delivery time.
- Expiration time, which indicates the time range when the event is valid.
- Maximum number of notifications that can be queued for a single consumer. This effectively places an upper bound that lessens the load presented by misbehaving consumers.
- Order policy, which specifies the order in which notifications are buffered for delivery.
- Priority of events.
- Reliability of event delivery
- Both event reliability and connection reliability. If fault tolerance properties are specified, the Notification Service reconnects to the set of clients and delivers all non-expired events to consumers after a crash or disconnection. At the message level: Best effort, persistent.

Furthermore, the event channel supports the following QoS properties:

- MaxQueueLength, which specifies the maximum number of events that can be queued.
- MaxConsumers, which specifies the maximum number of consumers that can be connected to the channel.
- MaxSuppliers, which specifies the maximum number of suppliers that can be connected to the channel.

Structured Events

The Notification Service defines a standard data structure for the events. The structured event illustrated in Figure 2.11 is a strongly typed event message that consists of a header and a body. The header contains two sections:

- the first stores fixed information, such as domain_name, event_-name, and type_name.
- The second section stores the variables and optional information about the event. This is a sequence of properties to hold QoS information related to the notification.

The body of the structured event stores the actual event data and is also divided into two sections:

- The filterable data, which is a sequence of properties. This part contains the fields that the consumers use to base filtering decisions on.
- The payload data.

The header and body are structured into two parts mainly because of performance reasons. When filterable data has its separate compartment, it is not necessary to touch the payload data upon filtering. Moreover, the notification could be contained within the optional header fields leaving the body empty. This would be even more streamlined.

Discussion

The centralized nature of the Event Channel as a CORBA object limits its scalability. All the registered consumers and suppliers are managed by the channel, which may limit the number of active entities and also

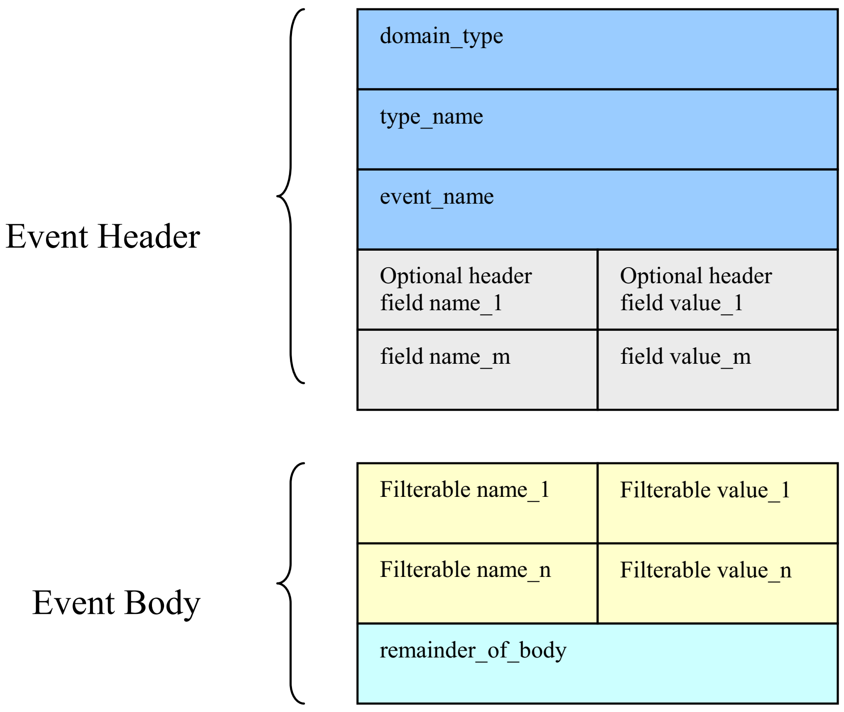

Figure 2.11: The structured event: Event header and event body.

the maximum number of notifications that the event channel is capable of processing in a given timeframe. Therefore it becomes important to create, manage, and specify federations of event channels. Each event channel has a master queue and a number of consumer queues. Each queue has some maximum capacity, which may be enforced using QoS policies supported by the specification. One way to relieve the bottleneck of the centralized event channel is to distribute these queues as CORBA objects, however, this kind of solution is still centralized. Since NS supports the federation of channels by connecting the supplier and consumer proxies, the system supports scalability.

Channel federation can be used to

- Improve performance by distributing consumers onto several event channels. Since an event channel is a CORBA object, it may become a bottleneck if the number of consumers (or producers) becomes large. Event channels may also be used to enhance local delivery by assigning to each event channel only local subscribers. In this case there is only one network invocation and a number of local invocations.

- Improve reliability by having multiple event channels for the same information. If one event channel fails, it does not necessarily prevent consumers from receiving the notifications.

- Improve flexibility by grouping consumers and producers into logical units (event channels).

2.3.6 CORBA Management of Event Domains

CORBA Event Service and Notification Service do not specify an event discovery service or a mechanism to federate event channels. Moreover, the procedure for connecting event channels is complex. The OMG Telecommunications Domain Task Force addresses these issues in the CORBA Management of Event Domains Specification [OMG01c], which specifies an architecture and interfaces for managing event domains. An event domain is a set of one or more event channels grouped together for management, and for improved scalability. The specification defines two generic domain interfaces for managing generic typed and untyped channels. Moreover, a specialized domain for both channels and logs is defined by the OMG Telecom Log Service specification.

The specification addresses [OMG01c]

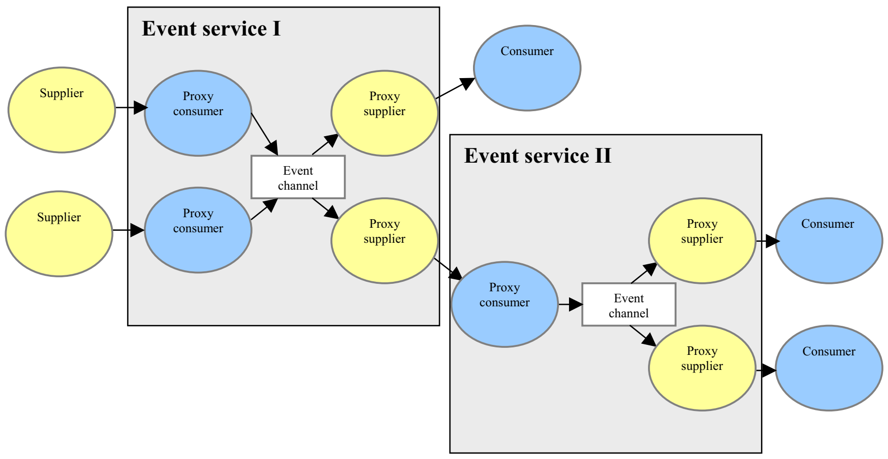

Figure 2.12: CORBA Notification Service channel federation.

- connection management of clients to the domain,
- topology management,
- sharing the subscription and advertisement information in an event domain, even when connections between event channels change at runtime,
- event forwarding within a channel topology, and
- connections between event channels.

It supports the creation of channel topologies of arbitrary complexity, allowing cycles and diamond shapes in the graph of interconnected channels. However, if events may reach a point in the graph by more than one route duplicate events need to be detected and removed. Moreover, if no timeouts are specified, events in a cycle will propagate infinitely. Therefore, the specification defines mechanisms that are used to detect cycles or diamonds in the network topology. Graph topology enforcement is done at channel connection time, and illegal connections are refused by the domain management.

Event suppliers inform the proxy consumers of event type changes using the offer_change callback. The channel is responsible for sharing this information with the consumers by executing offer_change on them. The consumer may be another channel and thus the change may propagate throughout the channel topology. Subscription changes work similarly, and the channel is responsible for invoking the subscription_change operation on all suppliers.

Event suppliers attached to the channel can obtain the types of subscriptions of event channels anywhere downstream by invoking obtain_-subscription_types on the proxy consumers. Similarly an event consumer can obtain the event types offered by suppliers on any event channel downstream by invoking obtain_offered_types on its supplier channels.

2.3.7 W3C DOM Events

W3C's Document Object Model Level 2 Events is a platform- and language-neutral interface that defines a generic event system [W3C00a]. The event system builds on the DOM Model Level 2 Core and on DOM Level 2 Views. The system supports registration of event handlers, describes event flow through a tree structure, and provides contextual information for each event. The specification provides a common subset of the current event systems in DOM Level 0 browsers. For example, the model is typically used by browsers to propagate and capture different document events, such as component activation, mouse overs, and clicks. The two propagation approaches supported are capturing and bubbling. Capturing means that an event can be handled by one of the event's target's ancestors before being handled by the event's target. Bubbling is the process by which an event can be handled by one of the event's target's ancestors after being handled by the event's target. The specification does not support event filtering or distributed operation.

The specification "An Events Syntax for XML" is a W3C Recommendation (11 October 2003) and defines a module that provides XML languages with the ability to integrate event listeners and handlers with DOM Level 2 event interfaces [W3C03g]. The specification provides an XML representation of the DOM event interfaces. The ability to process external event handlers is not required.

2.3.8 Web Services Eventing (WS-Eventing)

The Web Services Eventing (WS-Eventing) specification describes a protocol that allows Web Services to subscribe or to accept subscriptions for event notifications [BMT04]. An interest registration mechanism is specified using XML Schema and WSDL. The specification supports both SOAP 1.1 and SOAP 1.2 Envelopes. The key aims of the specification are to specify the means to create and delete event subscriptions, to define expiration for subscriptions, and to allow them to be renewed. The specification relies on other specifications for secure, reliable, and/or transacted messaging. The specification supports filters by specifying an abstract filter element that supports different filtering languages and mechanisms through the Dialect attribute. The filter is specified in the Filter element.

2.3.9 COM+ and .NET

Standard COM and OLE support asynchronous communication and the passing of events using callbacks, however, these approaches have their problems. Standard COM publishers and subscribers are tightly coupled. The subscriber knows the mechanism for connecting to the publisher (interfaces exposed by the container). This approach does not work very well beyond a single desktop. Now, the components need to be active at the same time in order to communicate with events. Moreover, the subscriber needs to know the exact mechanism the publisher requires. This interface may vary from publisher to publisher making this difficult to do dynamically (ActiveX and COM use the IconnectionPoint mechanism for creating the callback circuit, an OLE server uses the method Advise on the IoleObject interface). Furthermore, this classic approach does not allow filtering or interception of events [Pla99, Sri01, Mic02].

COM+ Event Service

The COM+ event service [Pla99, Mic02] is an operating system service that provides the general infrastructure for connecting publishers and subscribers. The service is a Loosely Coupled System (LCS), because it decouples event producers from event subscribers using the event service and a catalog for storing available events and subscription information. In this architecture, an event is a method in a COM+ interface called the event method, and it contains only input parameters.

The following steps are required to produce an event:

1. An event Class is registered.
2. Subscriber registers for an Event.
3. Publisher creates an Event Object at run time.
4. Publisher fires the Event by calling the method in the Event Object.

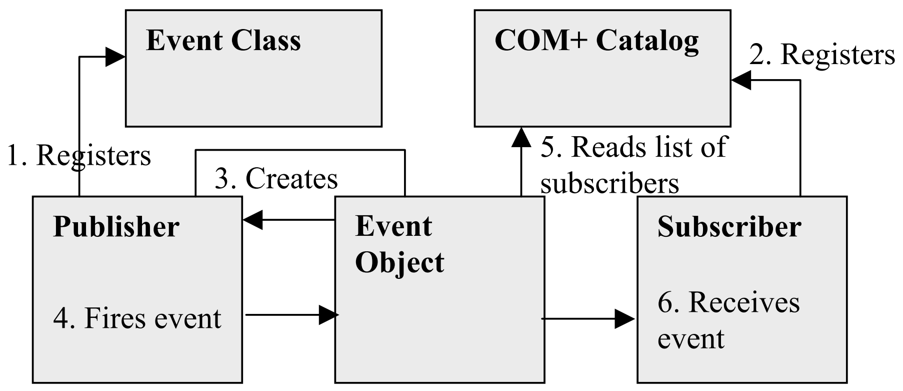

Figure 2.13: The COM+ Event Service.

5. Event Object reads the Subscription List from the Event Store.
6. Event Object delivers the Event to the subscriber by calling the appropriate method.

The change in the COM+ Event Service is the addition of the event service in the middle of the communication. The event service keeps track of which subscribers want to receive the calls, and mediates the calls. The event class is a COM+ component that contains interfaces and methods. A subscriber needs to implement the interfaces in order to receive the event, and a publisher calls the methods to fire events. Event classes are stored in a COM+ catalog that is updated either by the publishers or by the administration.

Subscribers register their wish to receive events by registering a subscription with the COM+ event service. A subscription is a data structure that contains the recipient, event class, and which interface or method within that event class the subscriber wants to receive calls from. Subscriptions are also stored in the COM+ catalog either by the subscribers or by the administration. Persistent subscriptions survive restarting the operating system whereas transient subscriptions will be lost on restart or reset.

The publishers use the standard object creation functions to create an object of the desired event class. This event object contains the event system's implementation of the requested interface. The publisher then calls the event method that it wants to fire. The event system implementation of that interface searches the COM+ catalog and finds all the subscribers who have expressed interests in that event class and method. The event system then connects to each subscriber, using direct creation, monikers, or queued components, and calls the specified method. Event methods return only success or failure. Any COM+ client can become a publisher and any COM+ component can become a subscriber.

The current event system has several limitations. The subscription mechanism is not itself distributed and there is no support for enterprise-wide repository. Secondly, event communication in the system is done either by DCOM or Queued Components, which are both one-to-one communication mediums. The delivery time and effort increases linearly with the number of subscribers, which means that the system is not scalable to firing events to many subscribers.

However, client-side disconnection is supported with queued components. COM+ supports components that record a series of method invocations (event occurrences) and are able to play them back in the recorded order. These components can be distributed using messages. Since the event object may be defined as queuable, a disconnected client may play back the desired event object upon reconnection.

COM+ Events can be extended to support filtering, which needs to be implemented either on the publisher side or on the subscriber side. If an event is filtered by a component on the publisher side, it is never delivered to the event service. If an event is filtered on the subscriber side the event service will make the decision of whether to deliver the event to a particular subscriber [Mic02].

Filtering on the publisher side is done by attaching a filter object to the event object interfaces (which correspond to events). The filter may query the subscription information and, for example, change the firing order for a set of subscribers. The subscriber-side filtering is done using parameter filtering for each subscription and method invocation. Parameter filtering evaluates the subscription FilterCriteria property against the parameters of the event method. The filter criteria string recognizes relational operators, nested parenthesis, and the logical keywords AND, OR, and NOT.

Interoperability with .NET

The COM+ Event System needs to generate some metadata in order to interoperate with the .NET world. However, an abstract definition of the Event Interface, Event Classes, and their attributes is needed [Kis01].

.NET

The .NET framework supports events at many levels. There is support for programming-language-level events and interoperability with COM events. The interoperation of Visual Basic .NET code and legacy COM component events is done using a runtime callable wrapper (RCW). In VB.NET listeners create event handlers, which are added to sources. The connection between events and event handlers is implemented by special objects called delegates. The benefit of the .NET runtime is that the events from components written in different languages, say C# and VB, are interoperable.

Microsoft's messaging infrastructure is called Microsoft Message Queuing (MSMQ) [Mic99b]. In this kind of architecture, applications receive and send messages using queues. MSMQ supports disconnected operation and is especially useful on intermittently connected Windows CE/PocketPC devices. MSMQ allows application writers to asynchronously send messages. MSMQ CE version can, for example, be used

- for messages transferred when in range (delivery tracking, quality control),
- for messages transferred once in a while (intelligent set-top boxes, inventory control, . . .), or
- when Producer and Consumer are not active at the same time.

MSMQ Product Architecture

MSMQ queues are either private or public. Public queues are stored in a directory service called Message Queue Information Store. Public queues are more expensive to use because directory access is not free. Moreover, Windows CE clients cannot host public queues. The CE MSMQ independent client can operate independently if the server is unavailable and store messages locally. The servers route and store messages and support clients in the form of a client proxy server and a queue manager. On the other hand, MSMQ supports also dependent clients that cannot store local messages and need the server. The architecture supports three delivery options. Fast memory-based reliable store-and-forward supports network loss, but not reboot, and cannot guarantee exactly-once semantics. Persistent guaranteed store-and-forward supports reboot, and persistent transactional message queuing guarantees exactly-once in-order delivery. Transactional guarantee at commit time is about delivery to the

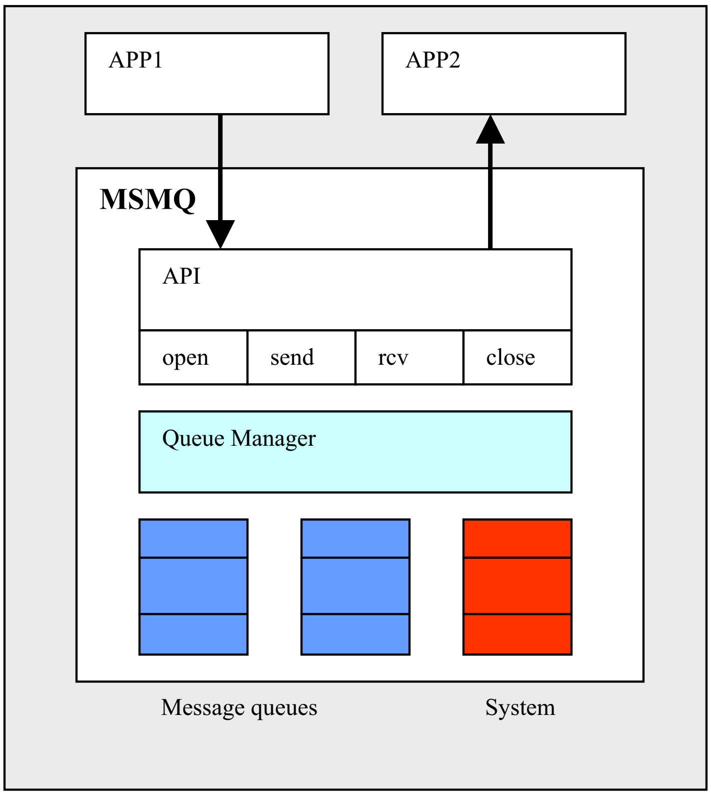

Figure 2.14: MSMQ Product Architecture. The Queue Manager connects to other Queue Managers in order to communicate between different hosts.

local queue. In essence, the system supports local all-or-nothing guarantee [Mic99b].

The MSMQ version for Windows CE (2.12+) supports roaming and dynamic adapter switching. It tracks Network Interface Cards (NIC) and restarts immediately after reconnection. The transparent storage is based on one queue per file. The footprint of the system is around 100-150K. The CE implementation has several limitations: clients must use direct names, only private queues are supported, the routing is limited, transactions are not supported (once and in-order are supported), there is no system support for encryption or ACL, and there is no remote queue access. The system can be deployed in a client-server or client-client environment and also for message-based IPC within a device.

The next version of MSMQ, Message Queuing 3.0, is available in Windows XP and supports messaging over the Internet, a one-to-many messaging model, and message queuing triggers [Mic02]. HTTP is supported as an optional transport protocol and an XML-based SOAP extension is introduced that defines a reliable end-to-end messaging protocol. By default MSMQ uses a proprietary TCP-based protocol. The system also supports real-time messaging multicast using the Pragmatic General Multicast (PGM) protocol [S+01]. This protocol supports only an at-most-once quality of service and does not support transactional sending. The MSMQ 3.0 programming model is extended to allow an application to send a single message to a list of destination queues.

Message Queuing Trigger is a service that allows an application to assign functionality in a COM object to be triggered when a message arrives in a particular queue. Each trigger is associated with a queue and applies a set of rules for every message arriving in that queue. An action is executed when all conditions in a trigger hold [Mic02].

Message routing is done using the lowest-cost route that is available. If a network fails, the next-lowest-cost route is used to deliver the message. Administrators define costs for each network with the management software (MSMQ explorer).

2.3.10 Websphere MQ

IBM's MQSeries, currently known as Websphere MQ, is one of the most popular MOM products for electronic business. The product supports heterogeneous any-to-any communication between 35 different platforms. MQ is compatible with JMS and integrates with Java Beans 2.0 (EJB), XML, and JSP framework and servlets. MQ also supports SOAP for Web service creation. A JMS 1.0.2 compliant embedded JMS provider supports point-to-point and publish-subscribe messaging [IBM02b].

MQSeries Everyplace enables access to enterprise data and supports mobile workers. Everyplace is available for a number of platforms, for instance Linux, WinCE, EPOC, and PalmOS. The PDA type messaging is similar to messaging for other platforms with queue managers. A queue manager manages queues that store messages, and applications communicate with their local queue manager. Remote queues are owned by remote queue managers, and each message that is inserted into a remote queue gets transmitted over the network. The queue manager may support a local queue, in which case the client is capable of supporting asynchronous communication. If no local queue is present, the client is bound to synchronous communication. Another configuration option is whether the client supports bridges and is capable of exchanging messages with other MQSeries queue managers.

A typical client-server configuration is a scenario where a server hosts the queue manager and clients connect to it with a bi-directional communication link (with a proprietary MQSeries protocol). The client infrastructure is quite lightweight, because it is dependent on the server queue manager. In a multi-server scenario, clients employ message channels, which support unidirectional, safe, and asynchronous message exchange. Channels are a form of end-to-end service provision and consist of the source queue manager, a number of intermediate managers, and the destination queue manager. The footprint of the system is 64K for Palm and 100K for a class file with Java devices [IBM02b].

2.4 Event Systems

This section presents event systems and prototypes. We present the Cambridge Event Architecture, Siena, Scribe, Elvin, JEDI, ECho, JECho, Rebeca, Gryphon, STREAM, and Rapide.

2.4.1 The Cambridge Event Architecture

The Cambridge Event Architecture (CEA) uses the publish-register-notify paradigm [BMH+00], in which the object publishes its interface, for example specified in IDL (Interface Definition Language, which is different from the IDL in CORBA). This interface includes the events of which it is capable of notifying. A client invokes the object synchronously and can register for events by indicating parameters (attributes) or wildcards. Wildcard matching is applied on the parameters of a notification, but it may not be applied on the event type. The template system provides rudimentary filtering by matching parameters one by one. The object accepts registrations and notifies the clients that match the registration template. The notification is performed when the event firing conditions and access restrictions are satisfied (Figure 2.15). The paradigm supports direct source-to-client event notification.

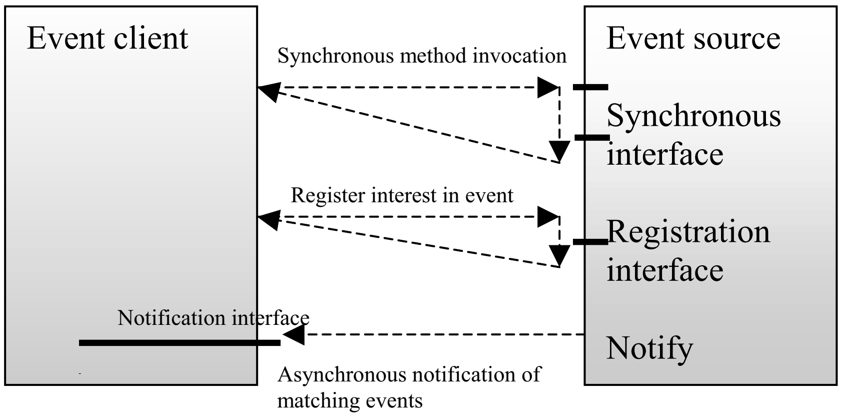
Event client
Synchronous method invocation
Event source
Synchronous interface
Register interest in event
Registration interface
Notification interface
Notify
Asynchronous notification of matching events

Figure 2.15: A publish-register-notify event architecture [BMH+00].

In CEA an object, if asked, publishes the events it is capable of notifying of in IDL. The object has a register method in its interface that has parameters for the type of event and wildcards. Event occurrences are objects of a specific type, and the set of types defines the level of event detection and notification granularity. CEA enforces access control upon registration, and authentication is based on a parameter value.

CEA supports defining intermediate services, which are called event mediators in the architecture. Event mediators act as middlemen between primitive event sources and event clients, and provide the facilities for detecting more complex events. Moreover, if an event source cannot afford the overhead of supporting template matching, it can send all its events to the mediator. The mediator then matches the template on behalf of the source.

The mediator is capable of providing equivalent functionality to the CORBA event service. The CORBA event service registers interests in all notifiable events with event sources and supports both a synchronous pull interface and an asynchronous push interface. Composite events can be detected by giving mediators the capability to filter simple events of different types across different sources.

The composite event detection functionality supported in CEA is a feature that is not present in many event systems. The event composition is supported by the combination of event templates. Composite events are detected by monitors, which are busy until the event is detected and fired. A composite event specification language may be used to design a monitor that detects complex templates. The system has been demonstrated by implementing an active badge system that monitors badges within a building.

Composite events have also been investigated at Cambridge [PSB03] recently. This paper presents a distributed framework for composite event detection and notification in a distributed environment. The system is based on JMS, and leverages the features of the underlying architecture. The key benefits of the proposed approach are the distribution of the detection task, an automata-based detection engine, and the use of an interval time model to detect the causality of events. A Lamport Logical Scalar Clock gives a causal ordering if such exists (but not a strict causal ordering). The paper presents a specification language for composite events. The system transforms the specification language to finite state machines. Formal semantics are given for the interval time model. The problem of translating non-deterministic automata to deterministic is not discussed other than to mention that the current implementation uses non-deterministic automata with a list implementation.

2.4.2 Scalable Internet Event Notification Architecture

Siena (Scalable Internet Event Notification Service) is an Internet-scale event notification service developed at the University of Colorado. Siena balances expressiveness with scalability and explores content-based routing in a wide-area network. The basic publish-subscribe mechanism is extended with advertisements that are used to optimize the routing of subscriptions [CRW99].

Several network topologies are supported in the architecture, including hierarchical, acyclic peer-to-peer, and general peer-to-peer topologies. Servers only know about their neighbors, which minimizes routing table management overhead. Servers employ a server-server protocol to communicate with their peers and a client-server protocol to communicate with the clients that subscribe to notifications. It is also possible to create hybrid network topologies.

Siena is similar to IP-multicast, however, the two mechanisms differ in the way they support groups of subscribers. IP groups are not very expressive. They partition the IP datagram address space and each datagram can belong to at most one group. Clearly, this creates problems if an event that spans several groups of subscribers is to be delivered.

Four different server topologies have been identified in Siena:

- Centralized
- Hierarchical
- Acyclic peer-to-peer
- Generic peer-to-peer

Naming and Filtering

Siena is implemented with a flat event namespace, i.e. event names have no structural correlation with each other. An event consists of a set of attribute-value pairs. Each attribute has a name and a value. Siena supports the types null, string, long, integer, double, and boolean.

A filter consists of an attribute name, a constraint operator, and a constraint. Siena does not support wildcards in the attribute name so the attribute names must match exactly to the names in the published event. A filter may include several filtering clauses, which are ANDed together. Thus every filtering clause or component must return true in order for the filter to pass the event. Siena supports the operators equal, less than, greater than, greater than or equal to, less than or equal to, string prefix, string suffix, always matches, not equal, and substring.

An example event:

```
string stock "abc"
int value     2.53
```

An example filter:

```
string stock = "cde"
int value > 1.0
int value < 1.5
```

Siena supports patterns, which are based on event attribute values and event combinations. A pattern is a sequence of filters that is matched to a temporally ordered sequence of notifications. Network latencies may cause some events to arrive in the wrong order, and these are ignored by the Siena solution.

Routing

In Siena, each event consists of a set of attribute-value pairs that are matched with filters. Each server on the event system routes events to other servers based on subscription information, advertisement information, and filters. Each subscriber may specify a filter to constrain the subscription. In the same fashion, each advertisement may also include a filter. Siena evaluates the filters and follows a policy where events are replicated downstream and filtered upstream. This means that events are replicated to the clients at the last possible moment, thus reducing the bandwidth needed to transmit the events. Upstream filtering means that events are filtered as close to the sources as possible in order to reduce the number of uninteresting events transmitted over the network. The simple filter syntax allows the decomposition of a complex filter into several more general filters, which can be evaluated upstream. A filter is only applied if it is less general than the one used in upstream.

The same principle of upstream filtering also applies to event patterns. Patterns are decomposed (factored) into elementary filters that are delegated to other servers. In the delegation process a server tries to assemble subpatterns that are delegable to other servers.

Siena uses covering relations to determine when a filter covers a notification, a subscription covers a notification, an advertisement covers a notification, or an advertisement covers a subscription. For example, subscription S1 covers S2 if it evaluates to true in every instance where S2 is true. Servers propagate the most generic subscription that covers a given set of subscriptions. This minimizes the downstream data structures, however, the complex computation cost is paid closer to the subscriber, because the subscriptions need to be matched and evaluated. The results of Siena indicate that the covering relations exhibit a complexity that is quite reasonable for a scalable service.

The Siena system supports two different notification semantics: subscription-based and advertisement-based. In subscription-based semantics subscriptions are introduced at every node of the event service and a notification is routed if it covers a subscription. In advertisement-based routing servers use the information provided by event producers to route incoming subscriptions. A subscription is only forwarded if it covers the advertisement.

Forwarding Algorithm

The forwarding algorithm that was developed in conjunction with the Siena project consists of a forwarding table and a set of processing functions. Conceptually the forwarding table is a mapping between predicates (sets of filters) and interfaces to neighboring nodes. Each predicate is a disjunction of filters, where each filter is a conjunction of elementary conditions. Each elementary conjunction must return true in order for a filter (and predicate) to map to an interface. Each filter may map to several interfaces [CDW01].

The forwarding algorithm iterates over the event attributes. It searches for a partial match from the set of filters, where a constraint belonging to a filter is matched by the given attribute. If the filter (with the partial match) is not yet associated with an interface, the algorithm increases a counter to keep track of matched constraints for the given filter. If the counter size is equal to the number of constraints in the filter, the filter is said to match. After processing one filter the algorithm checks if all filters are matched. The algorithm stops if either all attributes of the notification or all filters are processed.

The number of interfaces thus imposes an upper bound on the processing along with the number of attributes and filters. The forwarding algorithm is optimized using binary trees and lookup indices for attributes used in the filters.

The performance and scalability of the forwarding algorithm were demonstrated by running experiments with 1000 messages and various numbers of filters and other parameters. It was found that the algorithm has good absolute performance and good cost amortization over a variety of loads. The constraint index, which acts as a lookup table for attribute names over constraints, is used to quickly detect attribute names that have no matching constraints. If no attributes match the event can be discarded by the router.

The filter matching algorithm has recently been extended with several optimizations [CW03]. The algorithm uses a matching structure based on an index and selections over attribute filters. The paper proposes several enhancements, namely the selectivity table that is used to prune those predicates that cannot be matched.

Implementation

The current Siena implementation is a prototype that consists of Siena servers and client-level interfaces. The C++ version supports the peer-to-peer server and the Java version supports hierarchical servers. Currently, the C++ implementation is not compatible with the Java version. The Siena implementation uses TCP/IP for communication.

Simulation

The algorithms and topologies used in Siena were examined in a simulated environment. The hierarchical client-server architecture should be used when there is a low number of parties that subscribe and unsubscribe frequently. The acyclic peer-to-peer model was found to be more applicable to situations where the total cost is dominated by notifications and there are many ignored notifications [CRW99].

Current and Future Developments

Columbia University has developed the XML-based Universal Event Service (XUES) that consists of three main services that support event handling for the Kinesthetics eXtreme (KX) real-time monitoring architecture. The system inputs events using the Event Packager, analyzes events using the Event Distiller, and dispatches events using the Event Notifier. The system interacts with other event systems using XML, FleXML, and Siena.

During the development of the Siena-XML interface [Ere01] several problems with translating an XML-based hierarchical namespace to a flat namespace were identified and addressed. In the conversion process the nested structure of XML documents is converted into flat names that preserve the hierarchy by separating the hierarchies with dots. This is a typical way of describing hierarchical content; another would be to use the Windows or Unix file system notation. Now, a problem arises when there are duplicate elements in a hierarchy, which translate to an item with multiple values. Siena does not support this, and the Siena-XML interface currently ignores these duplicate values. One solution would be to include support for wildcards or multiple sets of values, for example simple list objects.

In the future Siena is envisaged to integrate at the network service level, coexisting for example with TCP/IP instead of working above the network level. This would eliminate an extra protocol layer, and provide greater efficiency in routing and forwarding. From the Siena viewpoint TCP/IP performs explicit address routing and Siena is based on content-based addressing. The risk in using Siena as a network service is that content-based routing is computationally more expensive than explicit-address or subject-based routing [Ros01b].

There is also work to make Siena support satellite-based wireless communication. Satellite-based communication has desirable properties for transmitting events, because routing is not necessary when the events are broadcasted rather than sent using point-to-point communication lines. Thus it is possible to notify large numbers of interested parties in one hop. However, wireless networking is more unreliable than wired networking. Moreover, the receiving devices may be different from desktop computers, thus requiring the solution to cope with limited resources.

Siena has also been used as a peer-to-peer network similar to Gnutella. The Java-based Quad uses the Siena prototype and supports query, advertise, and response. One of the differences between Quad and Gnutella is that with Gnutella the messages are propagated to all servers and filtering is performed by the provider at the last step. The main architectural difference between Gnutella and Quad is the separation of clients and servers. Thus the general advantage of peer-to-peer systems in dynamic networking is lost [Hei01].

Siena has been extended to support mobility and wireless clients. The mobility support involves a handover protocol that uses either subscription-based or advertisement-based semantics [CCW03].

One of the findings of the Siena project is that expressiveness and scalability are in conflict. Expressiveness is related to flexibility of notification and routing. Scalability, on the other hand, is about vast dimensions, heterogeneity, decentralization, and the use of resources.

2.4.3 Scribe

Scribe [CDKR02] is a topic-based publish-subscribe system that explores the scalability of the notification service in peer-to-peer environments. It is built on top of Pastry, which is a scalable, self-organizing peer-to-peer location and routing system. Scribe provides an application-level multicast system. Pastry is based on uniform ID keys that are used as host addresses. The system routes a message to the closest possible key. Scribe provides a best-effort notification delivery on top of Pastry and specifies no particular event delivery order. Moreover, Scribe does not support filtering, buffering, or mobility. The rendezvous point forms the root of a multicast tree. In other words, the responsibility for a given topic (group of subscribers) is hashed over the set of the servers. When a subscribe message is routed towards the rendezvous point, each intermediate node adds the previous node to its table of children. This information is used in the multicast protocol, which is similar to reverse path forwarding. Events may be published directly if the IP address of the rendezvous point is known. However, subscriptions need to be routed within the peer-to-peer topology. Access control can be enforced at the rendezvous point. Pastry can route around faulty nodes by resending the subscription and thus repairing the multicast tree.

2.4.4 Elvin

Distributed Systems Technology Centre (DSTC) has been developing the Elvin system since 1993 and it has grown from a single person research project to an effort with a team of programmers and researchers. Elvin is a general event notification service, which aims to improve on features identified in a 1995 survey of commercial event filtering software. Elvin started as a publish-subscribe notification service, but currently it is referred to as a content-based routing service. The Elvin team aims to standardize the Elvin protocol through the Internet Engineering Task Force (IETF), and the Elvin protocols are written in the style of IETF drafts. DSTC was a contributor to the OMG Notification Service RFP and one of the submitters of the CORBA Notification Service.

Elvin uses a client-server architecture in notification delivery. Clients establish sessions with Elvin servers and subscribe and publish notifications. An Elvin notification is a list of name-value pairs, similarly to that of Siena. Basic primitives are a 32- and 64-bit integer, a 64-bit double precision floating point, an internationalized string (UTF-8 encoded), and an array of bytes. Subscription expressions are defined using logical expressions with a C-like syntax: “stock == "abc" && value > 80”. The expressions are evaluated with Lukasiewicz’s tri-state logic that uses an additional value of indefinite (i.e. true, false, indefinite).

Elvin has language bindings for C, C++, Emacs Lisp, Java, Python, Smalltalk, and Tcl. Elvin is content-based, because it allows routing decisions to be made based on the whole message. Elvin features a decoupled security model, in contrast with the traditional point-to-point model, in which communication between publishers and subscribers is authenticated with keys. Producers and consumers can have overlapping key sets. This supports multi-party authorization.

Service discovery is done using a lightweight protocol that is based on multicast. Once a server has been deployed on the network, clients use the protocol to discover the server and dynamically register. Clients also listen to router advertisements, which are also distributed using multicast.

Elvin 4.1 was released on March 19th in 2003. This version includes web-based router management, configurable quality of service, support for automatic failover of standby routers, federation between routers, and new scalability support.

Clustering

Elvin supports local clustering of servers that improves scalability and distributes the local load. Clustering is used to implement a distributed, but single-subscription, address space. Routers within a cluster communicate using a reliable multicast protocol over an IP network. An Elvin router may force a client to reconnect to another server in order to reduce load. The Elvin cluster is similar in functionality to a web farm. An Elvin router is a daemon process that runs on a single server and distributes Elvin messages. Each router in an Elvin cluster shares client subscription information with every other node. Not all subscription information is shared, but only sufficiently in order for a router to decide if a given notification has any subscribers at any server.

The initial forwarding decision in server-server communication is done based on a list of terms. Messages are first analyzed at a local router and then multicast to the cluster. The set of destination routers is determined before multicasting by matching the message against the term list. Each packet contains the unique identifiers of the routers that have matching terms. This hasty approach results in a number of unnecessary notifications at the router level. The Elvin team aims to improve this in the next version of the system.

The Elvin cluster topology consists of a single master router and a number of slave routers. The master router maintains management data. All slave routers listen to management traffic within the cluster and keep information about every node. Routers also keep information about subscription terms of other servers, current states, the list of URLs offered by a router for client connection, and current router load and statistics. Master servers listen for join packets and keep track of the cluster as a whole. A new master router is elected using an election protocol if the old one fails.

Communication between clients and routers employs RPC-style communication with positive and negative acknowledgements. Delivery has best effort, at-most-once semantics. In the client-server protocol the server may drop notifications, but is obliged to warn the client that it has done so.

Federation

There is a different protocol for linking distributed clusters of servers to a federated system. The Elvin federation protocol assumes that the federated topology forms a spanning tree. Moreover, the linking protocol supports the definition of pull filters that constrain the notifications sent to other clusters.

Quench

In Elvin terminology quench means an operation supported by all event producers that gives the producers the possibility to evaluate a subscription expression to cease producing events that are no longer needed. Quench is also used to determine which notifications should be produced. In CORBA this would mean that the first event channel refrains from forwarding unnecessary notifications (CORBA does not support client side filtering). The quench is a semantic extension of the subscribe mechanism

In Elvin quench is implemented in the client-server protocol. Any client may request to be notified when the subscription status of the server changes. The client may request information on named attributes in subscriptions. The requested information is sent as an abstract syntax tree. There is also support for an automatic quench, which is implemented in the client library.

Mobile Users

Elvin has been extended to support mobile users. One of the requirements was persistence in order to keep undelivered notifications. Elvin is non-persistent by design so a prototype proxy was designed to store notifications. The proxy model extends the client-server architecture of Elvin by introducing the proxy as a third component. Proxies act as normal clients to servers, but as proxy servers to clients. In this design, clients connect to these proxies, which mediate the Elvin service [SAS01].

The proxy is able to handle multiple clients with separate sets of subscriptions. Elvin did not support subscription grouping by the client, so support for this was added to the system (the concept of a session). These sessions need not be client-specific, but may rather span multiple clients or applications. This stems from the observation that many people have several devices, but may wish to receive the same set of information regardless of the medium. In order to manage the storage space for undelivered notifications, the proxy supports the definition of a time-to-live (TTL) for each subscription. In addition, clients may specify the maximum number of notifications to keep.

In the current prototype clients explicitly connect to the proxy, and they must connect to the same proxy to retrieve notifications. Proxy discovery and roaming between proxies is not supported. The Elvin proxy service is proposed as a solution to proxy roaming and client migration between networks. However, the difficulty lies in that the proxy is a stateful entity, whereas normal Elvin servers are stateless.

Non-destructive Notification Receipt

For users who use many different devices and wish to share notifications, Elvin supports non-destructive notification receipt. This means that the proxy does not destroy a notification upon its successful delivery. Elvin ensures that notifications are never delivered to the same client more than once.

Because sessions may contain a number of clients, Elvin supports additional management functionality regarding the set of subscription set by clients. Each client is informed of the current subscription status. There may also be a number of sessions per client, in which case only one notification is sent even if there are multiple matches.

2.4.5 JEDI

Java Event-based Distributed Infrastructure (JEDI) is a distributed event system developed at Cefriel at Politecnico di Milano. In JEDI the distributed architecture consists of a set of dispatching servers (DS) that are connected in a tree structure [CDN01]. Each DS is located on a node of the tree and all nodes except the root node are connected to one parent DS. Each node has zero or more descendants. Event subscription and unsubscription requests are propagated by each DS upwards towards the root. Event notifications are processed similarly and forwarded by the local DS to its parent. Upon receiving an event, each DS checks its descendants if they have an interest in the event, and, if required, forwards the event down the tree.

This strategy requires that a given DS knows the event requests of its descendants in order to make the forwarding decision. Moreover, since all requests and notifications are propagated up the tree, the communication and processing overhead of the nodes near the root may become a bottleneck. If any of the nodes near the root become disabled, parts of the tree become isolated. In this case the system needs to deal with segmentation and to be able to mend the tree or negotiate a new root and a new tree.

A JEDI event is an ordered set of strings, the first string being the name of the event followed by event parameters. An Event Dispatcher can subscribe to a single event or an event pattern. Event patterns are used to filter events based on parameter matching, for example `foo(aa*,bb)` matches all events named `foo` that have exactly two parameters and the first parameter starts with `aa` and the second parameter is exactly `bb`.

JEDI preserves causal ordering of messages, that is, if event e_1 caused the firing of event e_2, e_1 must be delivered first to all interested subscribers. This mechanism allows a pair of components to synchronize through the generation of events [CDNF01].

The JEDI architecture is being extended to support mobile clients and ad-hoc configuration [CDNP00]. Publish/subscribe middleware makes a good candidate for utilizing context-aware computing. Asynchronous interest-based communication is a good start for building decoupled and adaptive software components. Compositionality and reconfigurability are being emphasized in JEDI and system supports mobility with moveOut and moveIn operations.

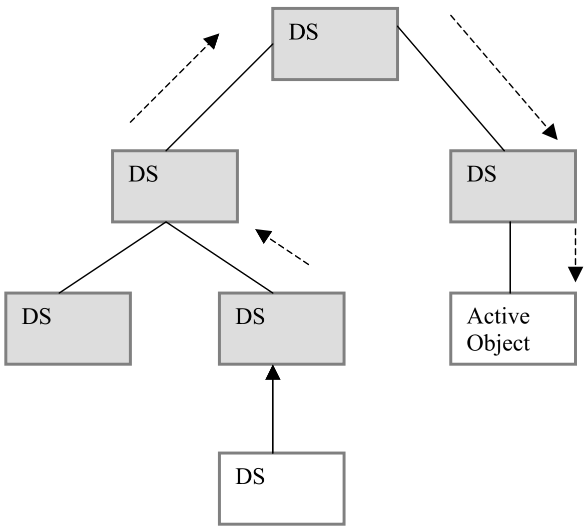

Figure 2.16: Event propagation in JEDI.

The dispatching servers in the JEDI architecture support mobility by allowing clients to disconnect, move to a new dispatching server, and connect while retaining all the notifications. The dispatching servers manage temporary storage for notifications. They also coordinate that no duplicates are received and that the notifications are causally ordered [CDN01]. The new dispatching server contacts the old one directly in order to receive the accumulated notifications. The old DS notifies its parent dispatching server to route any further notifications for this client to the new DS.

Notifications are routed in the JEDI dispatching tree from producers to consumers and there is no possibility for adapting the routing strategy to reflect changes in the pattern of communication. The system offers good performance if the tree is organized in a good way that minimizes network traffic. In essence, when clients migrate from one dispatching server to another the load placed on the servers changes. It may be necessary to recreate the dispatching topology to reflect these changes.

JEDI approaches the adaptation of publish/subscribe systems to more dynamic environments by extending the event routing mechanism with the addition of a new spanning tree routing algorithm. Now, a delegate leader is responsible for each subscription. The delegate accepts subscriptions of similar type and becomes the leader of the subscribers. It also manages the distribution of the group in the tree. Each dispatcher knows the group leaders for all subscriptions [CDNF01].

The JEDI approach is based on dynamically defining the dispatching tree by using approaches similar to multicast routing. The first strategy is to create a minimal spanning tree for each pair of publisher and group of subscribers, but this is considered to be inefficient. The second strategy is to have a single routing tree for each group of subscribers and have different publishers for the same class of events use the same tree.

JEDI uses a method called the Core Based Tree Strategy, in which the dispatchers are connected in a possibly cyclic graph and each dispatcher knows its neighbors. Dispatchers broadcast all unique subscriptions to all servers, and all subsequent subscriptions of the same type are sent to the party that sent the original subscription. The original source dispatcher has implicitly become the leader of a group of subscribers, and it maintains access to that group. Now, the source may balance load by assigning subscriptions to dispatching servers. All dispatchers know all group leaders, and those dispatchers that belong a group know the dispatching tree of that group. When a component unsubscribes, the associated dispatcher either leaves the group, continues to route notifications, or, if it was a leader, the system needs to elect a new leader for that group.

Mobility support in JEDI is still under consideration, for example the latency of updating the dispatching trees when clients are moving very frequently and in the case of abrupt disconnections are still open issues.

The current focus is on what kind of abstractions are needed at lower levels in order to detect disconnections at upper level. The scalability of the JEDI system to Internet-wide use is an open issue. JEDI was used to implement the Orchestra Process Support System (OPSS) workflow management system (WFMS) [CDNF01].

The JEDI subscription propagation algorithm was improved later by introducing advertisements. This new algorithm is similar to the Siena work, and covering relations are used to optimize routing. The impact of advertisements was evaluated using simulation, and the results show that with advertisements the root node spends much less time processing subscriptions. The simulation results on 8 to 85 dispatchers indicate that the processing time of advertisements is quite low (between 2.65% and 2.9%) [BDNFT00, BDNT00].

The JEDI project ended in 2000 and Cefriel has continued to work on event architectures. They have a project on fault tolerance and scalability issues in distributed communication based on the publish/subscribe paradigm. They continue to use the JEDI event dispatchers as a reference implementation. The goal of this research is to implement a fault-tolerant JEDI.

2.4.6 ECho

ECho is a high-performance data transport mechanism that is based on event channels [EBS01]. ECho uses channel-based subscriptions , similarly to the CORBA Event Service. ECho's derived event channel mechanism implements filtering by adding an application-supplied derivation function F to all listeners of a particular event channel, and by transferring all events that are generated by the sources and passed through the filters to a derived event channel. This scheme resolves issues in the delivery of unwanted events. ECho is especially optimized for streaming data and data transmission. ECho has been shown to perform better than Jini (distributed Java events), CORBA Event Channels, and XML-based messaging. ECho was developed at Georgia Tech and the source is available for academic research purposes.

2.4.7 JECho

JECho is a distributed event system that has been recently extended to support mobility using opportunistic event channels [CSZ03]. The central problem is to support a dynamic event delivery topology, which adapts to mobile clients and different mobility patterns. The requirements are addressed primarily using two mechanisms: proactively locating more suitable brokers and using a mobility protocol between brokers, and using a load-balancing system based on a central load-balancing component that monitors brokers in a domain. The mobility protocol is, in principle, similar to most mobility protocols (Wireless CORBA, Siena, Rebeca, . . .).

The filtering model is based on stateful user-defined objects, called modulators, which may transform the event stream. This allows more fine-grained filtering than non-state-based predicate matching. However, possible security problems are not addressed, and it may be difficult to do optimizations between similar modulators. In addition, client-based filtering is not addressed and it may also be difficult to implement efficiently. For example, a mobile producer should download all relevant modulators from the broker. Furthermore, no session management is provided so all user-specific modulators are relocated.

The system supports load balancing and resource monitoring, which are novel features for mobility-aware event systems. The paper presents simulation results for different scenarios, for example, relocation overhead and mobility patterns. Mobility patterns are examined in a 100-node network using BRITE and the evaluation includes scenarios such as random walk, salesman, pop-up, and fixed. Moreover, end-to-end delay and mobility/communication ratio are measured using a real system with two subnets.

2.4.8 Rebeca

Rebeca is a distributed event system that supports mobile users and context-aware subscriptions [FGKZ03]. The system supports both logical and physical mobility. The basic system is an acyclic routed event network using advertisement semantics. The mobility protocol uses an intermediate node between the source and target of mobility, called Junction, for synchronizing the servers. If the brokers keep track of every subscription, the Junction is the first node with a subscription that matches the relocated subscription propagated from the target broker. If covering relations or merging are used this information is lost, and the Junction needs to use content-based flooding to locate the source broker. A merging system was developed in the Rebeca project for conjunctive filters.

2.4.9 Gryphon

The Gryphon system was developed at the Distributed Messaging Systems group at the IBM T.J. Watson Research Center. Gryphon is a Java-based publish-subscribe message broker intended to distribute data in real time over a large public network. Gryphon uses content-based routing algorithms developed at the research center. The clients of Gryphon use an implementation of the JMS API to send and receive messages. The Gryphon project was started in 1997 to develop the next generation web applications and the first deployments were made in 1999.

Gryphon is designed to be scalable, and it was used to deliver information about the Tennis Australian Open to 50000 concurrently connected clients. Gryphon has also been deployed over the Internet for other real-time sports score distribution, for example the Tennis US Open, Ryder Cup, and monitoring and statistics reporting at the Sydney Olympics.

The Gryphon system supports both topic-based and content-based publish-subscribe, relies on adopted standards such as TCP/IP and HTTP, and supports recovery from server failures and security. In Gryphon, the flow of streams of events is described using an information flow graph (IFG), which specifies the selective delivery of events, the transformation of events, and the creation of derived events as a function of states computed from event histories.

Information flow graphs contain stateless event transforms that combine events from various sources, and stateful event interpretation functions that can be used to derive trends, alarms, and summaries from published events. Each event is a typed tuple. Stateful events depend on the event history. States are used to express the meaning of an event stream and the equivalence of two event streams.

The Gryphon model consists of information spaces, which are either event histories or states. Event histories grow monotonically over time as new events are published. Event sources and sinks are modeled as event histories. States capture certain relevant information about event streams, and they are typically not monotonic. Information spaces are defined using information schemas. Dataflows are directed arcs that connect nodes in the graph, which needs to be acyclic [BKS+99].

Gryphon supports four types of dataflows. Select is an arc that connects two event histories with the same schema. Each arc is a predicate on the attributes of the event type in the information space. All events that satisfy the constraint are delivered to the destination information space. The transform arc connects any two event histories that may have different schemas. Each arc has a rule for mapping event types between the two spaces. This rule may include functions that transform particular event attributes. The collapse arc connects an event history to a state using a rule. The rule maps a new event and a current state into a new state. The expand arc is the inverse of collapse, and links a state to an information space. When the state at the source of the arc changes, the destination space is updated in such as way that the sequence of events it contains collapses to the new state. This transformation is non-deterministic.

Gryphon has two techniques for the implementation of systems based on IFGs. The first is a flow graph rewriting optimization that allows stateless IFGs to be used with multicast technology. The second is an algorithm for converting a sequence of events to the shortest equivalent sequence of events.

The information flow graph is abstract and separated from the physical topology of the network. The mapping of an IFG to a network of message brokers is nontrivial. Gryphon reduces an arbitrary IFG by rewriting it. All the select operations are moved together and closer to publishers and all the transform operations are also grouped together closer to the subscribers. Transform operations are done at the periphery of the network.

The Gryphon system allows the representation of event histories as states, which is interesting especially for mobile and disconnected users. Wireless users would benefit if a system could inform them with a summary of events that occurred while they were disconnected (the state). The Gryphon system detects failed brokers and reroutes traffic around failed nodes. Moreover, the system incorporates several security mechanisms, such as access control, and four authentication methods.

Gryphon supports the JMS publish/subscribe API and topic-based subscription. In addition, clients may specify filters using the WHERE clause of SQL92 supported by JMS. Gryphon extends the publish/subscribe one-to-many model with request-reply and solicit-response models. By using unique topics JMS users can use request-reply-style messaging. In the solicit-response model a client may make an advertisement to which one or several clients may respond privately.

The basic unit of the Gryphon multi-broker configuration is the cell, which is a group of fully connected servers. Cells may be further linked together for geographical scaling through link bundles. Link bundles provide redundant connections between cells, which includes load balancing and fault tolerance not provided by gateway-based approaches. The internal protocols and systems ensure that cycles are avoided and messages are routed around failed nodes.

2.4.10 STEAM

The STEAM (Scalable Timed Events and Mobility) event system is specifically designed for wireless ad-hoc networks [MC03]. The system uses three different filters to address the problems related to dynamic reconfiguration of the network topology. Specifically, the STEAM system is intended for WLANs using the ad-hoc network model, and the main application domain is traffic management. The system uses an implicit event model in which entities subscribe to interesting event types locally, and not by using a centralized broker. STEAM exploits a group communication service for notifying interested entities. Groups are geographically bound and nodes are identified using beacons.

The three filter types supported by STEAM are subject, proximity, and content filters. Events consist of a name and a set of typed parameters. The name also determines the structure of the event. A subject filter is matched against the event and mapped onto a proximity group. A proximity filter corresponds to the geographical aspect of the proximity group. A proximity filter specifies the scope in which events are disseminated. A proximity filter applies to an event type and is established when the type is deployed. In essence, upon publication of an event the source matches the subject and proximity, and the subscribers match the content. This requires that the proximity filter at the producer must have location information from the subscriber.

The paper does not explain how this information is acquired, how often it is updated, and how the security implications are handled. In essence the protocol is a wireless application-level broadcast protocol with subject-based filtering at the source and content-based filtering at the client.

Producers announce the event types they intend to raise (publish) with the geographical area, called the proximity, within which events of this type are to be disseminated. The proximities may be defined independently of the physical range of the communication system. The routing layer may support multi-hop communication.

STEAM is based on the Proximity-based Group Communication Service (PGCS). In this service, groups are assigned certain geographical areas. A node that wants to join a group needs to be located in the group's area. STEAM provides a Proximity Discovery Service (PDS) that uses beacons to discover proximities. Once a proximity is discovered the associated events are delivered to the client if it has a matching subscription. PDS causes the middleware to join a group if either a subscription or an announcement matches the group. The proximities are static but clients may move.

An experimental scenario is presented in [MC03]: traffic lights at an intersection with experimental results with and without filtering. The results suggest that distributed filtering, although simple in this case, is beneficial in ad-hoc environments and may reduce the amount of transmitted traffic significantly.

2.4.11 Rapide

The Rapide language is designed to meet the requirements for architectural definition [LV95]. The main idea of Rapide is to use asynchronous events and their causal relations to model both static and dynamic architectures. In this context, an architecture consists of interfaces, connections, and constraints — an interface connection architecture. When the architecture specification is executed all causal relations are stored and checked against the constraints. The key requirements for the system were component abstraction, communication abstraction, communication integrity, dynamicity, causality and time, hierarchical refinement, and relativity.

The interdependencies of Rapide components are modeled using partially ordered sets. A pattern language is defined for detecting composite events. An event of a particular action is a tuple of information with a unique identifier, a timestamp, and dependency information. The system supports placeholders and universal quantification over types. Patterns are used in interfaces to define behaviors and in architectures to define connections.

2.5 Conclusions

Message-oriented middleware and event notification are becoming more popular in the industry with the advent of the CORBA Notification Service, the Java Messaging Service, and other related specifications and products from many vendors. Many research projects have addressed and are addressing issues of scalability, compound event detection, mobility, and fault tolerance, to name a few topics. There are many ways to classify event systems, and many possibilities for their use depending on the requirements.

Traditional MOM systems are getting influences from event-based systems. For instance, JMS supports both queues and publish-subscribe style communication with filtering. However, these systems usually lack support for distributed coordination in notification delivery, and they employ topic-based routing. Current event systems are evolving towards content-based routing, which uses the whole notification as an address. In content-based systems clients can change their interests without changing the addressing scheme (adding a new topic).

Scalability has been emphasized in Siena, and it has been designed for Internet-wide scalability and tested in a simulation environment with various network topologies. Wide-area operation introduces latency, which creates problems for notification semantics and mobility. Other systems address scalability and fault tolerance by creating clusters (Elvin) or cells (Gryphon) that contain connected servers. These clusters are connected using point-to-point links and possibly different protocols. Multicast and fault tolerance can be provided within the clusters. Event systems are logically centralized, however, the CORBA Event Channel is also physically centralized, creating a possible bottleneck.

Ad-hoc networks are emerging with the introduction of short-range radio communications. Ad-hoc event systems support the dynamic addition and removal of event servers (or event dispatchers). Ad-hoc event topologies are currently an emerging research topic and some research issues have been raised in JEDI. STEAM supports ad-hoc event dissemination and proximity groups.

From the mobile Internet and ubiquitous computing viewpoint JEDI and Elvin were one of the first systems to examine support for disconnected operation. JEDI supports both mobility and disconnected operation as a service and Elvin only disconnected operation (with a few additional features) as an extension to the original architecture. Almost all message queue products support disconnected clients with various semantics. One important decision is whether to include mobility support as an extension or as an integral part of the event service. If fault tolerance or mobility are to be supported, it may be necessary to integrate this functionality at the service level. Another open issue is whether the event service should reside at the network level or at the application level. For Internet-scale routing, as proposed in Siena, it might be beneficial to have some support at the network level. Siena has recently been extended to support mobile clients, and JECho and Rebeca also have a handover procedure for relocating subscriptions. Content-based routing and mobility is a challenging combination and most routed event systems that use covering relations or filter merging may need to use flooding in the mobility protocol.

Only a few architectures support complex compound event filtering. Usually event filtering is done using simple parameter wildcard matching (JEDI), simple clauses (COM+, Elvin), SQL (JMS, Gryphon), or Extended

TCL (CORBA). Compound event detection is supported in CEA with event templates and in Siena by detecting a sequence of simple filters. Compound event detection is also a feature that may be integrated as an external component or within the infrastructure.

Many systems do not consider the process of locating and connecting producers or of locating event channels (Notification Service). Some architectures, such as Siena, JEDI, and Elvin, support this within the infrastructure. There are two completely different problems: one is locating an access point using multicast or unicast to a known address, and another is to use either the infrastructure or some other service to locate channels or to subscribe. In systems such as Siena and JEDI, subscription is performed using the API and formulating a text-based filter. With CORBA it is necessary to obtain an event channel and go through a more complicated procedure in order to obtain references to proxy objects. This process of obtaining the event channel reference according to interests is not specified.

Many message queue products are now supporting XML-based solutions, such as SOAP, as one of the transport options. MQSeries, MSMQ, and .NET support SOAP, and Siena has XML bindings as well. XML has many applications in messaging and event-based communication. XML can be used to define the content of messages. For example, JMS facilitates XML-based messages and the routing of XML documents.

However, the building blocks of the semantic web, such as ontologies, are not yet supported. Ontologies and XML-derived languages could well be used to define events and event systems, and to improve interoperability. XML and a suitable ontology would enable the specification of complex event monitoring tasks that are uploaded to routers or, for example, web services.

Chapter 3

XML Protocols

Services provided over the Internet are becoming increasingly a major part of the current world. Currently the most often used system for implementing these services is a Web browser sending its service requests to Web servers, which then generate the responses dynamically. This system leaves much to be desired as anything more complex has to be done with add-ons such as cookies. Therefore the need for a more flexible and powerful system is obvious.

The client-server, Request-Response paradigm described above contributes much to the inflexibility of current service models. A new general architecture and a protocol to go with it are needed, if new service models are required. This system should also be simple to implement and provide enough flexibility to allow rapid development and deployment of various services.

We will review the concepts of XML and Web services and then concentrate on the protocol seen as the basis of these services. We will also discuss issues related to these new services in wireless environments and go through some proposed solutions.

3.1 XML

The standard way of marking up document structure on the World Wide Web (WWW) is Hypertext Markup Language (HTML) [W3C99a], which is based on Standard Generalized Markup Language (SGML) [ISO86] (the technical term for HTML is an SGML application). SGML is a framework for creating markup languages. This is done by writing a Document Type Definition (DTD), which describes the allowed markup tags and their syntax. SGML documents are typically hierarchical, i.e. elements (content between a start tag and its corresponding end tag) contain other elements in addition to text.

The intended way for an application to parse an HTML document is to implement a full-blown SGML parser, which uses the HTML DTD to parse each element according to its defined syntax. However, SGML allows DTD writers to leave certain parts optional, which is useful for both cutting the size of documents (e.g. by omitting end tags for specified elements) and for decreasing the ratio between markup and actual content. This makes writing an SGML parser difficult, and so WWW browsers typically implement only an HTML parser.

The World Wide Web Consortium (W3C), aware of the problems with HTML being an SGML application, set out to simplify SGML. The result of this simplification is now known as Extensible Markup Language (XML) [W3C04e] (the new version 1.1 [W3C04f] was prepared to address some issues in character encodings, specifically the changing nature of Unicode (`http://www.unicode.org/standard/standard.html`)). XML has no implicit content: all start tags must have a corresponding end tag. In addition, XML does not contain some rarely-used features of SGML.

```
<?xml version="1.0"?>
<message status="urgent">
  <from>Boss</from>
  <subject>Reports</subject>
  <text>
    I haven't yet received those reports you promised
    to deliver yesterday.  I need them ASAP.
  </text>
</message>
```

Figure 3.1: An Example XML Document

An example XML document is shown in Figure 3.1; tags are limited by `<>`, end tags begin with `/`. The part between a start tag and its corresponding end tag is called an element and the material strictly between them is called the element's content. Each element except the root (`message` in the example) is contained in another element, called its parent element. This contained element is naturally called a child element of its parent. An element may contain attributes inside its start tag (`status="urgent"` in our example). These attributes typically affect the processing of the element in some application-defined way.

As with SGML, specific markup languages can be created with XML using a DTD. However, DTDs are not seen as a good fit for XML so some alternatives to them have emerged. The most visible of these is XML Schema ([W3C01a] and [W3C01b]), a W3C Recommendation. XML Schema syntax is XML instead of the DTD language, so it seems to be a better fit for XML data. Also, XML Schema allows a more fine-grained and flexible approach to restricting element content. DTDs are still expected to last for a while, though, mostly due to their already-familiar syntax and established use base.

Other popular alternative schema languages are RELAX NG [OAS01] and Schematron [Jel02]. RELAX NG aims to be simpler than XML Schema with a strong theoretical grounding. Schematron is based on expressing constraints on the documents in XPath [W3C99d], which makes it more expressive than the other schema languages. RELAX NG is also an ISO (`http://www.iso.org`) standard, and Schematron is in the process of becoming one.

3.2 Web Services

The term Web services has in recent times risen to prominence. The point of Web services is to unite a large variety of different platforms into large distributed systems using simple, standardized protocols and interfaces. XML is an important component of Web services as they are realized today.

The definition of a Web service has not always been very clear-cut. However, there are certain overall characteristics that fit into all used definitions. The central one of these is the use of XML practically everywhere. XML is used to describe the service interfaces, to locate services, and even to encode the actual messages. XML is beneficial since it is flexible, standardized, and popular.

The W3C has also embraced the Web services area with its Web Services Activity (`http://www.w3.org/2002/ws/`), which was started in January 2002 as an extension to the XML Protocol Activity. This activity consists of the Description (`http://www.w3.org/2002/ws/desc/`), XML Protocol (`http://www.w3.org/2000/xp/Group/`), and Choreography (`http://www.w3.org/2002/ws/chor/`) Working Groups. The XML Protocol Working Group is the oldest of the three, started in 2000 as the XML Protocol Activity.

Originally, the Web Services Activity also included the Architecture Working Group, but their work was concluded by the publication of a W3C Note on the architecture of Web services [W3C04h]. Their work was not continued, since their purpose was seen to not fit very well to the W3C's overall mission. Their definition of a Web service is available in the glossary [W3C04i]:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

The XML Protocol Working Group has completed the basic Recommendations on SOAP [W3C03d, W3C03e] and is now working on some extensions. Of these, the XML-binary Optimized Packaging [W3C04j] and SOAP Message Transmission Optimization Mechanism [W3C04g] are at the Candidate Recommendation level. The Description Working Group's main Working Drafts [W3C04a, W3C04b, W3C04c] are based on the Web Services Description Language (WSDL) specification [W3C01c] and are currently at the Last Call stage. The intent of WSDL is to describe service interfaces at the level of individual messages or operations. The Choreography Working Group concentrates on a higher level than the other groups, attempting to define patterns of message exchanges in Web services and languages for describing these patterns.

Current integration work in the Web service area is largely coming from Microsoft and IBM, who publish many of their technologies as W3C Notes. Specifications related to Web services are also published by OASIS (`http://www.oasis-open.org`), who have concentrated on integrating existing Web service specifications and creating new ones based on existing business needs. The most important OASIS specifications in the Web service area are the ebXML message service [OAS02b], which specifies a messaging service built on top of SOAP, and Universal Description, Discovery, and Integration (UDDI) [OAS02a] for description and discovery of Web services.

It is expected that in the near future the number of mobile devices having a continuous wireless connection to a network will increase rapidly. Because of this, it is important to evaluate the various Web service technologies in light of the unique challenges offered by wirelessness and mobility. Of the three parts of Web services, protocol, description, and discovery, the most obvious part to concentrate on is the protocol since that one is most clearly affected by the move to wireless networking. In addition, if server components can be on a mobile platform, discovery will be more complicated than with fixed services, though this will probably be handled with special addressing schemes suitable for mobility rather than by changing the actual discovery process.

3.3 Protocols

In considering the protocols we will be concentrating on SOAP as defined by the W3C. However, the emerging issues are mostly related to XML use as a message format and our considerations should be applicable to other XML-based messaging systems.

3.3.1 History

The origins of SOAP lie with UserLand Software. Their Frontier product is a content management system for the WWW. It also includes a Remote Procedure Call (RPC) interface to more easily provide interconnections between various services on the WWW. For this RPC system, they designed a new protocol, XML-RPC [Win99].

XML-RPC is a minimal protocol, intended to be used only for simple RPC needs. There are provisions only for a single Request-Response round trip messaging, there are only a few basic datatypes, and the only transfer protocol is Hypertext Transfer Protocol (HTTP). These can also be seen as advantages of XML-RPC. Due to its simplicity, it is easy to implement and there are in fact dozens of implementations in several different languages. There are also no provisions for extensions so the specification has remained stable for several years.

Even though other communication patterns can be built on top of a RPC framework, this would require careful specification of interfaces and messaging semantics. These challenges led to the need for an extensible messaging system. Microsoft had expressed interest in utilizing XML-RPC in their future products, so they teamed with UserLand to produce such a system, which was named SOAP, for Simple Object Access Protocol.

SOAP gained popularity quite fast after its initial launch and other companies joined Microsoft and UserLand in developing SOAP further. Version 1.1 was published as a W3C Note [W3C00b] in May 2000 by Microsoft, IBM, Lotus, DevelopMentor, and UserLand. After this, SOAP was adopted by W3C's newly formed XML Protocol Activity for standardization. This activity, nowadays the XML Protocol Working Group in the Web Services Activity, has produced a Recommendation of SOAP version 1.2 [W3C03d, W3C03e] in June 2003.

3.3.2 Features

We will be concentrating on the features of SOAP version 1.2, since that is the current version, and the one to which existing Web service platforms are moving towards.

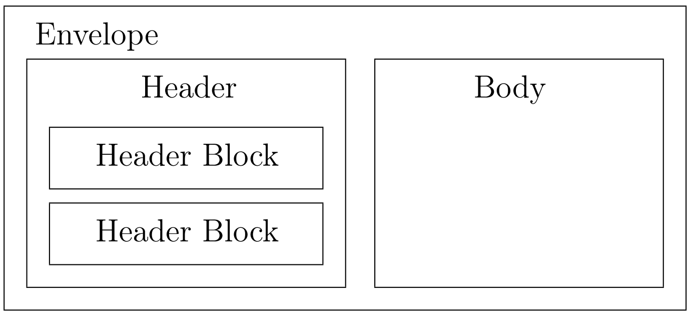

Figure 3.2: The Structure of a SOAP Message

The SOAP message structure is shown in Figure 3.2. A SOAP message is an XML document with the root element being Envelope. This element contains one or two elements, the first of these being an optional Header and the second one being a mandatory Body. The Header may contain any number of child elements, called Header Blocks. The SOAP specification does not address the content of the Body element in any way, other than requiring it to be well-formed XML and specifying its structure in the case of errors, so-called SOAP Faults. This leaves it to each application to define how the content of the Body element is to be interpreted.

The extensibility of SOAP stems from the fact that the content of the Header element is very loosely specified. There are practically no requirements on the type or content of individual header blocks. The only semantics that are defined for header blocks are a few optional attributes describing the encoding of the block, the intended recipient of the information in the block, and an indication of whether the block's semantics must be understood by the recipient. This loose specification allows application developers to freely define their own header blocks with appropriate semantics.

SOAP's extensibility allows it to be used in a wide variety of situations. There is in fact no necessity for SOAP messages to be responded to: the specification assumes only a one-way message transfer from the sender to a receiver. However, as the W3C's XML Protocol Usage Scenarios document [W3C02b] describes, by defining a few header blocks, SOAP can be used to support such communication patterns as Request-Response, RPC, Event Notification, and Conversation. In addition, there are provisions for independent intermediaries to be placed on the path between the initial sender and the ultimate receiver.

SOAP version 1.1 was still practically tied to HTTP as a transfer protocol. There was no other specified protocol mapping nor was there a suitable protocol framework to assist in using other protocols. This has changed in version 1.2, where the HTTP binding was removed to the Adjuncts section of the specification and replaced in the main part by a generic protocol binding framework. This framework should make it easier to use SOAP over non-HTTP transfer protocols, and in fact there already are implementations of SOAP that support other protocols. In particular, the XML Protocol Working Group has published a SOAP binding for email as a W3C Note [W3C02a] to illustrate the usefulness of the binding framework.

There was originally very little SOAP-specific work done on the security aspects of Web services. For a long time, the only way of having some security with SOAP was to use Secure Sockets Layer (SSL), as is done with HTTP. However, the Web services world is much more complex than the plain Web world and SSL does not address all relevant issues such as authentication in connection with intermediaries and third parties, or security after a message has reached its destination.

The security situation has been changing now that people have realized SSL is not sufficient for the needs of SOAP. The W3C is working on encryption, signatures, and key management in the context of XML, and has published Recommendations on signing [W3C02d] and encrypting [W3C02c] XML documents. This work has been used by OASIS to define a security solution for Web services [OAS04].

3.3.3 Current State

As mentioned above, SOAP is now officially being developed by the W3C. It is currently a Recommendation, which means that this version should be stable for some time and that implementations will probably aim for and advertise version 1.2 compliance.

The XML Protocol Working Group was originally chartered to be disbanded in April 2002. However, this timetable would have required the Working Group to publish a Candidate Recommendation in April 2001 and a Recommendation in September 2001. The current charter terminates in January 2005.

SOAP's roots in XML-RPC are also somewhat of a hindrance to full utilization of SOAP's features. XML-RPC is a RPC protocol over HTTP, as was SOAP in the beginning. However, the current version of SOAP is neither RPC- nor HTTP-specific. Even so, people often associate SOAP with these two concepts and this misconception also causes misunderstandings of the SOAP specification.

3.3.4 Implementations

There are several SOAP toolkits available for various languages. The most popular in the Open Source world are the `SOAP::Lite` module for Perl (`http://www.soaplite.com/`) and Apache Axis for Java (`http://ws.apache.org/axis/`). The kSOAP (`http://www.ksoap.org/`) toolkit is intended for Java Mobile Information Device Profile (MIDP) devices, and there is also an official specification for Web services on the Java 2 Micro Edition (J2ME) platform [Sun04]. Microsoft also includes its own SOAP toolkit in their .NET development framework (`http://msdn.microsoft.com/netframework/`). There are also SOAP bridges so that CORBA or COM objects can be exposed as Web services.

Interoperability testing of the various implementations is hampered by the fact that there is no good test suite. Therefore interoperability can only be expected in common cases and more obscure parts of SOAP probably do not get much interoperability testing. The XML Protocol Working Group has published a Recommendation [W3C03f] containing a set of tests for SOAP implementations.

After the gradual stabilization of the SOAP 1.2 specification, implementations have been targeting conformance to this version, and most popular ones advertise themselves as such already. The XML Protocol Working Group used to keep a partial list of 1.2-conformant implementations and their statuses, but this list was not kept up-to-date, probably due to the stabilization of the specification and the improved conformance of implementations.

3.4 XML over Wireless

If XML Protocols are intended for use in the services of the future, the needs of mobile users must be taken into account. Mobile users are typically behind low-bandwidth high-latency wireless links and the protocols and data formats originally designed for wired networks may be too heavy for wireless connections. In addition, devices such as Personal Digital Assistants (PDAs) and mobile phones typically have weaker processors, and users would like power usage to be as small as possible to conserve battery, so processing requirements are also a significant factor.

3.4.1 Problem Areas

There are several problems in trying to use SOAP over a wireless connection as noted in e.g. [LH03] and [KTR03]. The most obvious of these is that XML documents tend to be quite large, since the tag names are usually quite descriptive and XML does not allow certain redundant information to be left out. Another problem is that the typical underlying transfer protocol in SOAP implementations is HTTP according to the standard SOAP binding, which might not be suitable for typical wireless environments.

XML is also fully text-based, which means that processors need to do string matching and conversions of integers and floating-point numbers between text and binary representations. This can be quite costly in terms of processing power, which is a scarce resource in the context of small devices.

3.4.2 Transfer Protocols

The only transfer protocol specified by the W3C for SOAP is HTTP, which is practically always used over TCP. However, TCP is not very well suitable for wireless links [BPSK97] and HTTP itself is somewhat heavy. Of the implementations, `SOAP::Lite` for Perl supports several protocols other than HTTP including FTP, raw TCP, and Jabber, an instant messaging protocol. Most implementations have a protocol framework of some form, which is supposed to make switching from one transfer protocol to another easy.

An application protocol framework called Blocks Extensible Exchange Protocol (BEEP) was published by the IETF in March 2001 as a Request for Comments (RFC) [Ros01a]. BEEP is a peer-to-peer protocol that supports connection sharing between logically separate sessions. Implementations of BEEP (`http://www.beepcore.org/beepcore/projects.jsp`) already exist for Java, C, and some other languages. In some circles, a suitable BEEP-based protocol is seen as a possible replacement for HTTP.

A standardized SOAP binding for BEEP has been published by the IETF as a RFC [OR02], but there are still no implementations, and very little interest in this. There was also interest in mapping BEEP on top of Stream Control Transmission Protocol (SCTP), which is expected to be a popular transport-layer protocol in wireless environments, but this interest seems to have faded, and no specifications or implementations are available.

3.4.3 Compression

Compression of XML documents sent over the network would seem to be the method that gives the largest payoffs. There are three different ways to approach compression: non-XML-specific methods, methods taking advantage of XML's inherent structuring, and binary XML, a subset of the previous one that preserves the document structure even in compressed form.

Generic Compression

Generic compression algorithms can naturally be used also for XML documents. They are to be expected to perform well due to XML documents being text and the element parts being highly repetitive.

Generic compression is already publicly available in SOAP implementations. For example, the `SOAP::Lite` module for Perl implements transparent deflate compression using the zlib compression library (`http://www.gzip.org/zlib/`). In addition, the Apache web server has an extension module (not SOAP-specific) for zlib compression of served documents. It is in fact common these days for Web browsers and servers to support compression of documents sent with HTTP.

Typical compression algorithms achieve good compression ratios by exploiting redundancy in the data. From this it follows that they perform better on larger documents. While XML in general might be used for even very large data collections, individual SOAP messages are often quite small, typically a few kilobytes at most. Therefore the compressor may perform badly, possibly leaving the compressed size to over 50% of the original size [GS00, KTR03].

XML Compression Methods

Since XML is as popular as it is, it is to be expected that XML-specific efforts are also made, in compression as in other fields. XML-specific compressors typically exploit the additional structure present in the data. Usually these compressors can also exploit DTDs and XML Schema definitions of the document structure to achieve even better compression.

Some high-performance XML-specific compressors are XMill (http://www.research.att.com/sw/tools/xmill/), XMLPPM (http://www.cs.cornell.edu/People/jcheney/xmlppm/xmlppm.html) and XMLZip. None of these are in very active development, and XMLZip is not even available any more. All of them achieve very high compression ratios, but mostly for larger documents, and they all use methods for generic compression, which makes their processing time requirements quite large.

A specific type of XML-specific compression is binary XML, which is different enough to merit separate treatment. In binary XML tags are replaced by binary tokens, reducing each tag to one or two bytes. Standard attributes of elements can also be tokenized in this way. The benefits of using a binary encoding also manifest themselves even on shorter messages, like those used in SOAP. In addition, it is also possible to compress the content of elements independently of the tag compression.

Another benefit of this tokenization is that the document can be parsed directly from the compressed form without having to uncompress first, since the original structure remains intact; only the tags are changed. It is also possible for the message sender to generate binary XML directly. These could be beneficial since handling strings, which regular XML requires, is more time-consuming than handling pure binary data.

The best-known and oldest format of binary XML is WAP Binary XML (WBXML). This was published by the WAP Forum [W3C99b] for the needs of Wireless Application Protocol (WAP), which needs to be suitable for small devices with wireless connections. In WBXML the tokens are divided into code spaces and during encoding/decoding there is a default code space at each point in processing. The Millau [GS00] work expands upon WBXML with some enhancements of the format and performance measurements.

Binary XML is starting to become a more mainstream concept than it was a few years ago. The W3C hosted a Workshop on Binary Interchange of XML Information Item Sets[1], the purpose of which was to determine whether it is feasible for the W3C to start standardization work on binary

[1]http://www.w3.org/2003/08/binary-interchange-workshop/Report

XML formats. As a result of this workshop, the W3C formed the XML Binary Characterization Working Group (http://www.w3.org/XML/Binary/), the task of which is to gather information on situations where the overhead of XML is unacceptable, and to provide guidelines on determining whether alternate serialization formats are usable. The Working Group has published a Working Draft of a Use Cases document [W3C04d], which identifies use cases where XML deployment is currently problematic for some reason.

Currently several different binary formats for XML data are available. Apart from WBXML, which is e.g. supported by the kSOAP toolkit, newer ones are BiM (http://www.computer.org/proceedings/dcc/1477/14770467.pdf) used in MPEG-7 [AS01b], the XBIS Encoding (http://xbis.sourceforge.net/) supporting XML Infoset [W3C04k], and Cubewerks CWXML (http://www.cubewerx.com/main/cwxml/), a new Application Programming Interface (API) for XML with support for binary encodings.

There is also other recent work in the area of Web service optimization. Sun Microsystems (http://www.sun.com/) has gone the furthest in this direction with Fast Web services [SPGK+03], which maps the XML infoset of a SOAP message to Abstract Syntax Notation One (ASN.1) (http://asn1.elibel.tm.fr/). This technology is going through the process of ITU-T (http://www.itu.int/) standardization.

3.5 Conclusions

Judging from industry support, the deployment of Web services is about to increase quickly in the near future. With increasingly more sophisticated and powerful mobile devices coming to market, the number of users in wireless environments wanting to use these services seems likely to also grow rapidly. Reconciling the protocol overhead of Web services with the still quite limited data transfer capabilities of wireless devices is therefore very important.

One option would of course be to do nothing and accept the overhead of Web services as a natural part of communication, even in the wireless world. This attitude reputedly works for some people. The obvious benefit would be that there would be no need to implement a separate solution for wireless devices. Instead, any Web service implementation would suffice for both the wireless and the wired worlds. This option should be kept in mind as a baseline due to its simplicity, but tests should be done comparing it to various proposed improvements.

Switching to another transfer and lower layer protocols could be used with ordinary SOAP intermediaries that understand the transfer protocols. There would in this case be no need to touch the messages except insofar as SOAP intermediaries normally do while processing a message. The wisest course here would be to identify solutions that are expected to become popular, so that there would be fewer problems with support. Currently there seem to be no real alternatives to sending XML over HTTP.

Generic XML compression schemes should also be ignored as they typically perform worse than more generic compression on SOAP messages. Here the deflate compression method, which is expected to be supported quite widely, should be kept as a baseline, and proposed other schemes compared with it. Some form of binary XML would seem like a better alternative and, with the current interest in it, will probably feature in the future of Web services in the mobile Internet.

Chapter 4

Synchronization

4.1 Introduction

In this section we review work relevant to the Mobile Distributed Information Base (MDIB) work package of the project. We focus on the data synchronization aspects of storage systems suited for XML storage with characteristics such as high availability, consistency, and support for weakly connected and disconnected operation.

We use the definition of data synchronization given in [BP98]. According to the definition, data synchronization consists of the following subtasks:

Update detection determining which objects in a collection have been changed and need to be synchronized.

Update propagation the method by which updates are propagated between the locations that are synchronizing.

Reconciliation the synchronization logic that deals with the possible occurrence of concurrent data modifications.

In the reviewed work, synchronization is usually not the main research topic. Typically the research concentrates on something that entails synchronization as an integral part, such as a distributed file system or a shared database. The only work that specifically deals with synchronization as reviewed here is the SyncML synchronization protocol.

Although it would be interesting to review work that concentrates on synchronization, it is certainly beneficial to view it as an integral part of a system, as the described synchronization methods in these cases solve an actual synchronization problem. The holistic perspective also helps to recognize different aspects of the synchronization process, such as

- Policies regarding when, what, and how to synchronize.
- Conflict resolution mechanism, including conflict resolution policies.
- Low-level network transportation method (e.g. FTP, SSL).
- Caching and maintaining cache coherency.
- Consistency guarantees for synchronized data.
- Locking and session semantics.
- Methods for gathering and transporting updates (e.g. change log, deltas).

Throughout this document, we have tried to describe these aspects, to the extent they are applicable, for each of the reviewed systems.

The review focuses on the synchronization mechanisms in distributed storage systems and how suitable these are for operation in a mobile environment. Particular points of interest on synchronization from a mobile perspective are

- Are unexpected disconnections handled well?
- Does the protocol save bandwidth?
- Is the architecture suitable for peer-to-peer[1] operation?
- Is the protocol or architecture too complex for mobile devices?
- Is the architecture secure?

The distributed storage systems we review here are Coda, Bayou, OceanStore, and InterMezzo. Coda and InterMezzo are distributed file systems, whereas OceanStore is a highly available and fault tolerant data storage facility designed to be deployed on a global scale. Bayou is a distributed database, designed from the ground up with disconnections in mind.

We are also interested in how easily these systems could be utilized in the Fuego Core project. We have tried to evaluate each system from this aspect as well, by looking at such issues as

[1] In this chapter, peer-to-peer operation is assumed to mean end-to-end communication between IP-enabled devices without the need for supporting server infrastructure

- Is the system documented well?
- What is the API provided to application developers?
- What is the status of the implementation?
- Is there any source code available?

In addition to storage systems, we review the SyncML synchronization protocol as an example of a synchronization protocol that is not tied to a particular application and is supported by a range of mobile devices.

Furthermore, we present key-based routing briefly in section 4.5, investigate synchronization policies in section 4.8 and the reconciliation aspect of synchronization in section 4.9. The use of different encoding methods to minimize bandwidth usage is presented in section 4.10, which focuses on the use of delta encoding. We present our conclusions in section 4.11.

4.2 Coda

The Coda file system (`http://www.coda.cs.cmu.edu`) [BBHS00, SK92, MES95, Sat96, Bra98, S^+90], originating at Carnegie Mellon University and building on the heritage of the famous Andrew File System, is perhaps the best-known file system with support for weakly connected and disconnected operation. Since its initial development during the years 1990–91, and subsequent enhancement for weak connectivity in 1993–95 it has been extensively studied and refined. As research software Coda is quite mature. Current development concentrates on allowing Coda to be widely deployed.

The features of main interest in Coda are

- Support for disconnected operation
- Support for weakly connected[1] operation with adaptation to available bandwidth
- Free and relatively mature source code available
- Support for write access to shared file systems in disconnected mode
- Ability to ensure the availability of important files during disconnected operation

[1]High-latency, low-bandwidth (typically 9.6–64kbps) connections with occasional involuntary disconnections, typically wireless links.

- Client and server software are available on several platforms.
- Excellent compatibility with legacy applications

In addition, Coda supports replicated file servers for higher performance and fault tolerance, server recovery, and client authentication and access control using a Kerberos-like scheme combined with access control lists.

The architecture of Coda follows the client/server paradigm. The design is optimized for access patterns exhibiting few or no concurrent writes to the same objects. On the server side, Coda stores files using its own scheme, meaning that you cannot just "start sharing" an existing file system.

Coda exhibits a hierarchical architecture. At the highest level of organization, there is the Coda *cell*. The clients and servers in a cell share the same common namespace and configuration information. Each cell has a server designated as the System Control Machine (SCM), which is responsible for maintaining the configuration databases. Movement of clients between cells is currently not possible, so a cell would typically contain the entire shared file tree of an organization.

The Coda namespace, which appears as a directory hierarchy under a mount point (typically `/coda`) is populated by *volumes*. Volumes are subtrees of a server directory hierarchy, typically larger than a single directory but smaller than an entire partition. For instance, a home directory `/home/ctl` on server A could be exported to the Coda namespace as `/coda/home/ctl`. A volume may be exported inside the directory structure of another volume.

Volumes may be replicated across several servers for fault tolerance. The group of servers replicating a volume is called a Volume Storage Group (VSG), and the group of servers in the VSG available to a client at a given instant is referred to as the client's Active VSG (AVSG).

Security in Coda was designed on the basis that servers are trusted, whereas clients are not. For authentication and authorization during a session, Coda clients use a token obtained from a server in exchange for the correct password. Permissions are granted based on looking up the user who owns the token in the system's Access Control List (ACL).

4.2.1 Storage and Update Model

In this section, the operation of the Coda client and server is described first, after which the interaction between a client and a server is examined.

Server-to-server communication is not described in detail, as our main interest lies in the mobile aspects of Coda.

On the client side, Coda makes aggressive use of caching, not only to improve performance but mainly to make files available in states of weak connection or disconnection. The Coda client, consisting of a relatively small kernel-level module and the cache manager Venus, has three main responsibilities:

- Managing the cache: checking currency of files in the cache, fetching files, and making sure that the files the user has selected for off-line availability are in the cache.
- Propagating changes from the client to the server. Changes include modifications to files, directories, and permissions.
- Maintaining a log of changes to the file system when changes cannot be propagated immediately. This log is called the client modification log (CML).

The server part of Coda, called Vice, performs the following tasks:

- Manages server storage. File data is stored on the file system provided by the OS. Coda metadata, such as volume and directory information, is stored in a transaction-enabled raw partition to provide fault tolerance.[1]
- Grants and executes callbacks (in Coda terminology “breaks”) to clients when an object is modified.
- Applies CMLs received from clients.
- Performs conflict detection and resolution.
- Handles server-to-server replication; initiated by clients when they detect stale data on a server.

Coda uses session semantics for shared files. The session starts when a file is opened and ends when the file is closed. Coda treats files as atomic objects, meaning that concurrent modifications at different locations in a file will result in conflicts. Consistency and recoverability of Coda metadata is provided through the use of the recoverable virtual memory (RVM) [MS91] transaction handling module.

[1]Replaced by a binary file on Windows.

Hoarding, Emulating, and Write-disconnected

The interaction between Venus and Vice takes place in three states: *hoarding*, *emulating*, and *write-disconnected* (this state was called the *reintegrating* state before support for weak connectivity was added). These states correspond to being connected to a fast network (hoarding), disconnecting and working on the road (emulating) with occasional weak connectivity (write-disconnected) and finally returning back to the home network (hoarding). The states and possible transitions are depicted in Figure 4.1.

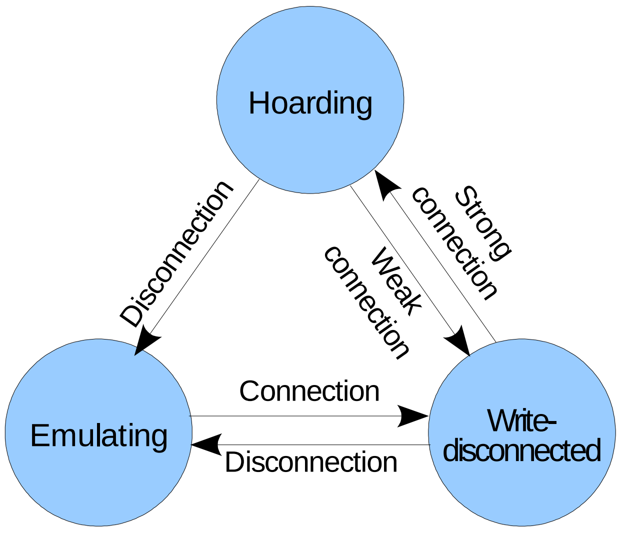

Figure 4.1: Venus states and state transitions.

In the hoarding state client changes to directories, files, and permissions are immediately propagated to all servers in the AVSG (replication is thus primarily handled by clients sending their updates to all servers in the VSG) in addition to the locally cached copies. Each client chooses[1] a *primary server* in the VSG, from which it fetches current objects and receives callback notifications ("breaks") when objects on the server are updated. The callback notifications are thus used to mark cached objects invalid. A client alerts the primary server to initiate server-to-server replication if it detects that any of the members in the AVSG has an old version of an object. Object currency is detected through the use of version vectors [P+83].

[1]Selection techniques include random selection or selection based on server load.

When Venus detects that the client has been disconnected from the network, it enters the emulating state. Ideally, the application user is not affected at all by the disconnection, as Venus tries to emulate connected operation in this state (hence the name of the state). The ability to continue using the file system as if nothing had happened during disconnection is a feature pioneered by Coda.

In disconnected operation we can no longer be assured that the objects in cache are up-to-date, and the penalties for a cache miss are fatal — the file cannot be accessed at all. To prevent vital files from being absent from the cache, the user is able to specify a list of files, called the *hoard database,* that should always stay in the cache (so-called “sticky” entries)[1]. As the modifications to the client cannot be propagated they are stored in the CML, which is replayed on the servers in the AVSG once the client has reconnected. As in the hoarding state, modifications are still applied to cached entries immediately. To save resources (and bandwidth at reconnection) the CML is subjected to optimizations, e.g. entries describing the creation, writing, and subsequent deletion of an object (the life cycle of a temporary file) are purged from the CML.

When the client has reconnected to the network (either over a weak or strong link), Venus enters the write-disconnected state, and the process of reintegrating the changes between the server and client starts. As modifications may have occurred on the server side, the cached entries on the client may be stale. Furthermore, to propagate the modifications local to the client the CML needs to be sent to the AVSG.

As the connection may be weak, bandwidth usage during reconnection must be considered. Overlooking this fact made early incarnations of Coda unbearably slow when reconnecting over a weak link. Fortunately, with the introduction of rapid cache validation, trickle reintegration, and user-assisted miss handling, performance over weak connections improved enormously. These techniques are described below.

To minimize cache validation traffic, version stamps for volumes as well as individual files are maintained. Version stamps for volumes enable validation of a large amount of files in a single sweep, provided no modifications have occurred to the volume, as is frequently the case. Using volume version stamps enables rapid cache validation.

Trickle reintegration is a process whereby the client continues to generate updates to the CML instead of sending them directly to the AVSG, even though the device is connected. The CML is allowed to age for some

[1]Files from the hoard database can still be evicted from the cache, if space is insufficient.

time, enabling optimizations to be done before it is sent to the AVSG. For instance, if the CML is allowed to age 10 minutes before being transmitted, a file create/delete pair 9 minutes apart can be optimized away. To increase responsiveness, an upper bound is set on the size of the transmitted CML chunks. The tradeoff of trickle reintegration versus hoarding is weaker consistency.

As opposed to fully disconnected operation, cache misses can be handled in the write-disconnected state. In case of a low-bandwidth connection, the delay experienced by the user when fetching large objects may, however, be prohibitive (a cache miss for a 1 MB file on a 9.6kbps link will cause a delay of some 20 minutes). User-assisted miss handling means that in cases where huge transfer times would result, Coda will query the user before initiating the transfer.

Reintegration and Conflict Handling

Reintegration, which takes place constantly in the write-disconnected state and when entering the hoarding state, reconciles the differences between a client and the servers. Reintegration consists of the following steps:

1. Venus makes final allocation of resources tentatively obtained from the server.

2. The CML is transmitted to the AVSG, which executes the CML operations, checking for conflicts at the same time. Some conflicts can be solved automatically (such as adding of files to the same directory), but not all: for instance, if a file has been modified on the server since disconnection as well as on the disconnected client, the conflict cannot be automatically solved. In such cases, it is possible to have Coda invoke *application-specific resolvers* to handle the conflict.

3. Updated files are fetched from the client.

If the CML causes a conflict, the log is rejected and the corresponding entries flushed from the client cache. In this case the user must resolve the conflicts manually.

4.2.2 Coda as an Mobile Distributed Information Base (MDIB)

When it comes to support for mobility, Coda is well thought out. There is support for weakly connected operation, including unexpected and spurious disconnections. The weaknesses of Coda lie in the requirements not specifically related to mobility: it is not designed to scale to a global level, since there are no guarantees for strong consistency (ACID) or support for transactions.

It should also be noted that the design of Coda, especially the authentication and authorization system, is based on the client/server model. An interesting question is if Coda could be extended to support operation in peer-to-peer mode.

4.2.3 Practical Issues

The sources for Coda are publicly available, and the system has reached a high level of maturity for research software. The semantics are easy to understand, lots of documentation is available, and the storage API is familiar to Unix programmers. In short, Coda should provide an excellent platform for experimentation.

On the downside, Coda uses its own partition format on the server side and requires a fair amount of configuring to work, raising the threshold for spontaneous experimentation and deployment.

4.3 InterMezzo

InterMezzo (`http://www.inter-mezzo.org`) [BN99, Bra02] is a newcomer to the family of distributed file systems with support for disconnected operation. The goal of the InterMezzo project is to achieve the same benefits as Coda as well as performance close to a local file system, but with a simpler architecture.

The architecture of InterMezzo was heavily inspired by that of Coda and several researchers have been involved in both projects. Like Coda, InterMezzo makes use of aggressive caching and has session semantics based on file open and close. The update propagation scheme is similar to that of Coda's weakly connected operation. The main differences are that InterMezzo utilizes the underlying file system to a higher degree (no special partition required on the server) and that many performance optimizations unrelated to networking have been performed, such as moving code to kernel space and introducing asynchronous calls between the InterMezzo modules.

InterMezzo consists of two modules:

1. The kernel file system code, called Presto, which gathers a log of modifications to the file system. This log is called the kernel modification log (KML).

2. The cache manager, which is responsible for keeping the cache updated and for sending the KML to the system's peer.

The cache manager originally consisted of a single program called Lento. Lento, which can still be used, has since been superseded by a HTTP-based approach consisting of a generic web server (such as Apache) and a program called InterSync. In the following discussion, we will assume that InterSync (with an HTTP server) is used. Lento essentially does the same as InterSync combined with a web server; the differences are mainly in how communication is initiated. The use of a web server for communications automatically adds support for proxies and secure transfers to InterSync.

4.3.1 Storage and Update Model

InterMezzo is a filtering file system [HP94] that sits on top of an existing journaling file system capable of supporting nested transactions, such as Linux's ext3 and ReiserFS. The underlying file system stores files in the same directory hierarchy and with the same names as those seen in InterMezzo. Some additional special files and directories are also added for control purposes. InterMezzo relies on the journaling abilities of the underlying file system to handle recovery and guarantee consistency. The advantage of this design is that it leverages the performance and transaction handling capabilities of an existing file system (unlike Coda which uses its own transaction handling system, the RVM).

InterMezzo can be set up for both one- and two-way synchronization. In the former case, changes are only propagated from the server to the client. In this case it is sufficient for the client to use InterSync without having an InterMezzo partition, since no modification log needs to be captured on the client.

In the case of two-way synchronization, changes need to be tracked on the client as well, and thus the shared files must reside on an InterMezzo partition. Two-way synchronization can be performed in two different ways: one is to use a web server on the server side only, in which case the client KML is pushed to the web server, the other is to have web servers on both the client and the server. In the latter case, depicted in Figure 4.2, the client and server operate symmetrically. We will examine this case more closely.

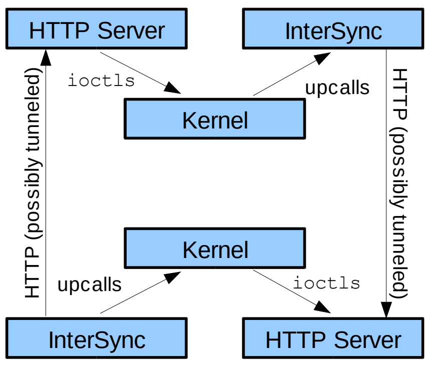

Figure 4.2: Symmetric two-way synchronization operation in InterMezzo [Bra02].

Modifications on the server side are propagated to the client by having InterSync fetch the KML (through a normal HTTP file request) regularly from the server. InterSync then processes the KML, fetching the corresponding file from the server whenever a file modification record is encountered. Modifications on the client are propagated similarly by having the server download the KML of the client.

As an alternative to polling the KML, synchronization may also be initiated by a message from the server whenever modifications occur. Furthermore, InterSync can be configured to only fetch the modified files when they are actually accessed on the client. Some optimizations are done when processing the KML, such as not fetching temporary files and avoiding to fetch the same file multiple times.

The scheme described above raises the question of how a server handles the KML when synchronizing to several clients (with different times of previous synchronization). The answer is that the server stores the last successfully retrieved record of the KML for each client and only sends more recent records of the KML to the corresponding client. Unbounded growth of the KML is prevented by having it periodically truncated. To be able to restore a heavily out-of-date client, InterMezzo maintains another log, the *synchronization modification log* (SML), which only contains object creation records.

Presumably, one can use server-granted write permits (callbacks), such as those used in Coda, to guarantee a higher degree of consistency. We were not able to find detailed descriptions of the permit handling.

Conflict detection is handled by checking the KMLs for conflicting operations. Assume that the KMLs gathered since the point of synchronization of directory trees are L_1 on the client and L_2 on the server. The client receives L_2 in order to perform reintegration. We need to check for possible conflicts between the logs L_1 and L_2, such as modifications to the same file.

Conflicts are automatically resolved according to a given policy by generating new logs L_1^{local}, L_2^{local} and L_1^{remote}, so that $L_1^{local}L_2^{local}$, when applied on the client, yields the same result as applying $L_2L_1^{remote}$ on the server (L_1^{remote} is the KML sent to the server). The following policies are mentioned:

Mobile policy and High-availability policy The conflicting object is kept on the client. When a conflicting server object is detected, the client object is moved away to another location.

Re-synchronization policy Used for heavily out-of-date systems, which need to apply the synchronization modification log (SML).

The details of generating L_1^{local}, L_2^{local} and L_1^{remote} can to some extent be found in [Bra02]. The need for L_2^{local} and a L_1^{local} differing from L_1^{remote} arise due to the fact that conflict resolution mechanism may require different operations to take place on the server and client.

4.3.2 InterMezzo as an MDIB

InterMezzo implements basic support for disconnected operation, and can thus operate in a mobile environment. However, there is no explicit support for weakly connected operation as there is in Coda. Noting the large improvement in performance that was achieved in Coda by accounting for weak connections, one suspects that InterMezzo could be improved as well. Techniques that come to mind are protocol optimization and delta transfers.

InterMezzo was designed for simplicity, which should be advantageous in mobile environments. Especially the layered file system design and KML gathering seem efficient and elegant, provided that an underlying journaling file system is available. On Linux-based platforms, this should not be a problem; there is even a journaling file system for flash devices available: JFFS2 [Woo01].

High availability and scalability have been considered to some extent, but cannot be compared to the massive approaches taken in e.g. OceanStore (see section 4.6).

Although the basic design of InterMezzo is client/server, it exhibits a great deal of symmetry between these. This speaks for easy adaption to peer-to-peer operation. Compared to Coda, the ability to use InterMezzo “on top” of an existing file system is a clear advantage: one is not forced to set up new partitions and the risk of data loss due to a file system bug appears to be considerably smaller.

4.3.3 Practical Issues

Sources and documentation for the InterMezzo project are available from the project web site as well as SourceForge (http://sourceforge.net/projects/intermezzo), where open-source development of the file system takes place. There is some testimony that InterMezzo has reached a level of maturity where it can be used on a day-to-day basis [vH02]. The sharing semantics are familiar from Coda and the API is the standard Unix file API.

On the downside, we note that there are only a few documents on InterMezzo available. The best way to learn the details of InterMezzo appears to be to read the source code. The use of kernel code will make deploying on platforms other than Linux harder.

As for security, InterMezzo leverages standard Unix and HTTP mechanisms for handling authorization and authentication. The choice not to build a security mechanism of its own should help keep down the number of security-related bugs.

Project development activity seems to have declined[1]during 2003. The latest official release is still 1.0beta1, released in November 2000, and there have been no new documentation or announcements of new developments.

4.4 Bayou

The Bayou (http://www.parc.xerox.com/csl/projects/bayou) [TD+94, E+97, PST+97] storage system was designed with collaboration of frequently disconnected users in mind. In Bayou, the database used by a collaborative application is aggressively replicated to provide high availability. To facilitate work in off-line mode Bayou abandons the requirement

[1]2002: CVS commits 520, mailing list posts 157 (devel), 339 (discuss], 4 (announce) 2003: CVS commits 18, mailing lists posts 91 (devel), 275 (discuss], 1 (announce).

for strong consistency among replicas and instead introduces a number of different guarantees for weakly consistent operation.

New data may be written to any available replica. From there changes will eventually propagate to all replicas by means of the update propagation mechanism, which in Bayou is *anti-entropy*. A very interesting idea in Bayou is the bundling of write checks and merge procedures with database updates.

Other important features of Bayou are

- use of committed and tentative data (as in OceanStore),
- different session guarantees for weakly consistent data,
- building on the relational database model, and
- probable suitability for peer-to-peer operation due to the anti-entropy update mechanism and easy replica insertion and deletion.

The project was carried out during the years 1993–97 at the famous Xerox Palo Alto Research Center (PARC). During the course of the research project several collaborative applications were built on top of Bayou, including a group calendar and an email application.

4.4.1 Storage and Update Model

A *write* in Bayou terminology is a procedure that generates a set of updates to be applied to the database. A write may for instance delete a row in a relational table. The storage system at each replica consists of a log of writes and the database that results from applying these writes in order. In theory, the write log on each server contains all writes to the database, received either from clients or other replicas (in practice, writes that have been committed can be discarded from the write log). The task of the update mechanism is to reach an eventual agreement among all the servers on the set of writes in the log, as well as the order of the writes.

When a server receives and accepts a *client write* it assigns the write an *accept stamp*, and associates the server ID with the accept stamp. The accept stamps are assigned in a monotonically increasing fashion, and they define an ordering for all the client writes received by a specific server: if write A is accepted before write B, A will precede B in the ordering. This ordering is called the *accept ordering*. The accept ordering is maintained in the write log.

When receiving a write from another server, the write already has an accept stamp, which is left unmodified. Each server stores the last received accept stamp in a version vector, indexed by the server that assigned the accept stamp. For instance, if the version vector on a server S were $\{S : 1000, S_1 : 2002, S_2 : 3003\}$, it would mean that the accept stamp of the latest client write on S was 1000, and that S has so far received client writes of S_1 up to accept stamp of 2002, and client writes of S_2 up to 3003.

Using these concepts, we can present the basic operation of the anti-entropy update propagation mechanism. Write operations are propagated between pairs of servers. A server S propagates the writes it has received by contacting a randomly chosen replica R, and asks R for its version vector V. S then iterates through its write log, sending any write from a server s with an accept stamp a to R if $V[s] < a$. That is, S only sends writes to R that R has not seen previously. The pseudo code for basic anti-entropy is shown in Figure 4.3.

```
anti-entropy(S,R) {
    Get R.V from receiving server R
    # now send all the writes unknown to R
    w = first write in S.write-log
    WHILE (w) DO
        IF R.V(w.server-id) < w.accept-stamp THEN
            # w is new for R
            SendWrite(R, w)
        w = next write in S.write-log
    END
}
```

Figure 4.3: Basic anti-entropy executed at server S to update receiving server R [PST+97].

Using this scheme, all writes will eventually reach all replicas, according to the theory of epidemics. Also note that the accept ordering allows compact representation, in the form of version vectors, of all the set of writes seen by a server.

In practice Bayou exhibits some modifications to the basic anti-entropy scheme:

- Writes are divided into *committed* and *tentative* writes. One database replica is designated to be the *primary replica* and it determines a total ordering of writes by assigning monotonically increasing commit sequence numbers (CSNs) to all the writes it receives.

On all servers, the writes that have been assigned a CSN are committed. The committed writes are ordered before any write without a CSN in the write log. Writes without CSNs are tentative writes.

To accommodate for this modification, each server R maintains a counter of the highest CSN assigned to a write. During anti-entropy, the sender S checks if it has a higher CSN, and in that case sends the CSNs of all writes between $R.CSN$ and $S.CSN$ to R, along with any writes unseen by R. $R.CSN$ is then updated to $S.CSN$.

- The introduction of committed writes allows Bayou servers to truncate the write log. Any write with a CSN may be removed from the write log, as it is known that its position in the log will no longer change and hence its effect on the content of the database has been determined. The tradeoff is that if anti-entropy needs to be performed with a server beyond the log truncation point, the entire database needs to be transmitted.

- Anti-entropy can be performed through transportable media. The update log from a given starting point can be easily exported, e.g. to a CD-ROM and played back on a number of replicas. The receiving replicas just ignore any updates they have seen before.

- Writes are causally ordered. To provide the session guarantees presented later on, a causal write order is introduced, which allows determination whether, at the time of write B to a server S, another write A was known to S. This can be implemented with a modification to accept stamp assignment that still preserves the condition of monotonic increase.

- Server creation and retirement are light-weight operations. Bayou uses special writes (that propagate as normal writes through anti-entropy) to indicate server creation and retirement. To accommodate for this, each server needs to support version vectors whose size can be dynamically adjusted.

The write operations in Bayou were designed to be very flexible to account for the loss of strong consistency. Each write operation has three parts: a dependency check, an update set, and a merge procedure.

The update set consists of updates, insertions, and deletions to the database. The dependency check specifies the conditions that must hold in order to apply the update set to the replica. It consists of a generic structured query language (SQL) query and the expected result of the query.

A write passes the dependency check if the query returns the expected result when executed on the replica. The dependency check is thus used to detect conflict situations.

If the dependency check fails, the merge procedure part of the write is executed. The purpose of the merge procedure is to provide alternate courses of action when the update set could not be applied. In other words, the merge procedure can be used to handle conflict resolution, but it may also defer it, e.g. by writing the offending record to an error log.

As an example, consider the write to a group calendar in Figure 4.4. The update set of the write tries to reserve the conference room for Kevin

```
Bayou_Write(
   update = {insert, Meetings, 12/18/95, 10:00am, 60min, "Project Meeting: Kevin"},
   dependency_check = {
      query = "SELECT key FROM Meetings WHERE day = 12/18/95
         AND start < 11:00am AND end > 10:00am",
      expected_result = EMPTY},
   mergeproc = {
      alternates = {12/18/95, 12:00pm};
      newupdate = {};
      FOREACH a IN alternates {
         # check if there would be a conflict
         IF (NOT EMPTY (
            SELECT key FROM Meetings WHERE day = a.date
             AND start < a.time + 60min AND end > a.time))
               CONTINUE;
         # no conflict, can schedule meeting at that time
         newupdate = {insert, Meetings, a.date, a.time, 60min, "Project Meeting: Kevin"};
         BREAK;
      }
      IF (newupdate = {})   # no alternate is acceptable
         newupdate = {insert, ErrorLog, 12/18/95, 10:00am, 60min, "Project Meeting: Kevin"};
      RETURN newupdate;}
)
```

Figure 4.4: A Bayou write for a group calendar [E+97].

from 10 to 11am on December 18th. The dependency check of this write is to verify that the conference room is indeed empty at this time. The merge procedure tries to reserve the conference room at an alternate time and if that fails, it notifies Kevin by inserting the failed reservation in an error log.

[E+97] and [TD+94] present a number of session guarantees for weakly consistent replicated data, implemented in the Bayou project. These are

Read Your Writes ensures that reads within a session see any previous writes within that session. Without the RYW guarantee, a deleted message could reappear during a mail reading session.

Monotonic Reads guarantees that successive reads will see an increasingly current view of the database. That is, if an application has seen the effects of a set W of writes, the set of writes seen by subsequent reads, W_1, is guaranteed to be larger than W: $W_1 \supseteq W$. Without the monotonic reads guarantee, you might see a document in a directory listing, but get a “document does not exist” error when opening it.

Writes Follow Reads makes sure that traditional write/read dependencies are preserved in the ordering of writes at all servers (i.e the guarantee holds across sessions). WFR entails constraints on write operations with respect to ordering and propagation; if one of these is relaxed we get the WFR Ordering and WFR Propagation guarantees. Without the WFR guarantee, you could see responses to an article in a newsgroup before seeing the original article.

Monotonic Writes states that within a session writes must follow previous writes. As an example need for monotonic writes, consider storing a library and subsequently the application that uses it in a Bayou database. Without monotonic writes, you might see the application, but not the library.

4.4.2 Bayou as an MDIB: Practical Issues

Bayou was designed with disconnected operation in mind, and hence it works quite well in that respect. Anti-entropy handles disconnections well as the process can simply continue from the last received write when the connection is re-established. Furthermore, only a small amount of state is needed before the transmission of writes can start. No optimizations have, however, been made for weakly connected operation. As stated in [PST+97], quite a lot of bandwidth could be saved by optimizing the write messages propagated during anti-entropy.

The fact that each mobile device may work as a server raises some concerns regarding security, as one cannot naively trust every mobile server to be friendly. As discussed in [S+97], some level of security can be achieved by digitally signing each write operation and maintaining a trusted server site that keeps a full log of operations. The overhead of essentially having to sign every message is, however, not negligible and will impact performance.

Still, given that Bayou was designed with quite similar objectives as the MDIB under consideration in Fuego Core, it is very important to ask why it is not in widespread use, i.e. why has Bayou not been adopted as a storage solution for mobile devices?

The answer appears to be related to the difference between academic and practical usage, as Bayou was very successful as a research project and is one of the must-know references for any researcher in distributed storage.

When examining Bayou from a programmer/engineering point of view, there are indeed some issues which make the adaptation of Bayou harder:

- Using a non-standard API for storage. To store data, you need to use the Bayou API. The API also exposes complexities of the distributed storage as well as assumes a relational database model for data storage.
- The Bayou model of epidemic propagation of data between mobile devices may not be realistic. Most current mobile devices synchronize to some sort of "server" (home workstation, company server, etc.) in a client/server fashion. This model may also be easier to understand for the end user as well as the developer.
- Merge procedures, which in effect also need to implement distributed data validation, are not easy to write except in simple cases [SS02, sec. 7.4].
- The design requires an underlying database as well as an interpreter for the merge procedures, which may be an issue on constrained devices. One should also notice that the hierarchical file system, not databases, is by far the most commonly used storage solution in personal systems.
- No source code or binaries for Bayou appear to be available for download. Deploying a Bayou-like system would most likely involve rewriting our own implementation from scratch.

4.5 Key-based Routing

One of the central enablers for massively scalable, fault-tolerant storage is a Key-based Routing (KBR) infrastructure. The KBR infrastructure provides a fully decentralized way to route messages to *keys*, which are distributed over a set of *nodes* by some globally known function [DZD+03].

For a network of n nodes, current KBR algorithms generally perform lookups using $O(\log n)$ messages between the nodes, with an additional storage of $O(\log n)$ at each node. The additional storage is due to tables for routing lookup requests to the node storing the requested key.

Interest in KBR algorithms and distributed hash tables (DHTs), which are closely related to KBRs, have arisen with the advent of Peer-to-peer

(P2P) applications, as KBRs provide a suitable platform for truly scalable implementations. For instance, a trivial implementation of a P2P file sharing service such as Gnutella (`http://www.gnutella.org`) would assign each file a key and store the file data at the node of its key.

In the mobile context, dynamic nodes, i.e. nodes constantly joining and leaving the KBR, as well as the intra-node bandwidth consumed to maintain the routing tables become important. This issue is investigated in [RGRK03].

Several KBR/DHT algorithms and implementations have been developed during the last five years: CAN [RFH+01], Chord [SMK+01], Pastry [RD01], Tapestry [ZKJ01], and, more recently, Bamboo [RGRK03], which specifically addresses the issue of dynamic nodes.

4.5.1 Plaxton-Rajaman-Richa KBR

To illustrate KBR, we have chosen the algorithm originally proposed by Plaxton et al. [PRR97]. Assume a name space of b^k nodes. Each node N in the network is assigned a unique address in the range $[0, b^k - 1]$. When routing to a destination node D, the address of the node (which we will also denote with D, since there is no risk of confusion) is divided into k blocks $b_{k-1} \dots b_0$ so that $D = \sum_{i=0}^{k-1} b_i b^i$, with each block in the range $[0, b-1]$. For instance, using $b = 16$ and $k = 4$ the blocks of the address AB09 (base 16) would be 9, 0, B, and A.

Each node N maintains k neighbor maps with b entries, where the entry (i_k, i_b) contains the route to the closest node C (including N itself), whose i_k:th block equals i_b and the blocks $i_k - 1 \dots 0$ match the corresponding block in N.

Routing to a destination node D is done one block at a time, starting from block 0. Assume i is the current block. The next hop from a node N is given by looking up the node at position b_i in the i:th neighbor map, i.e. the entry (i, b_i). In this way, the destination is solved digit by digit. The maximum length of the path is trivially k (but usually less, since several neighbor map entries are loopback entries). For a routing example, see Figure 4.5.

Locating an object O_1, located at a server node S works as follows. A hash function is used to calculate a mapping from O_1 to a node R, which is the “root node” of the object. When O_1 is created at S it is published to the infrastructure by sending a message $\langle S, O_1 \rangle$ from S to R, which states that O_1 can be found at S. All nodes on the path from S to R (including R) store this information. When addressing O_1 from a node C we send a message

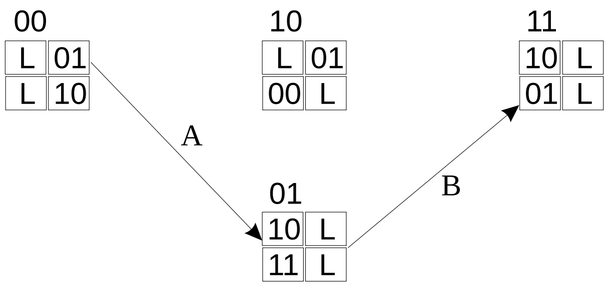

Figure 4.5: Routing using the algorithm of Plaxton et al. In this example $b = 2$, $k = 2$ and the distance measure is the geometrical distance between the nodes in the figure. The address of the node is written over the node's routing table. In the routing table, the entry (i_k, i_b) is at row i_k column i_b; L means that the entry points back to the node (a loopback entry). The arrows A and B show the routing of a message from node 00 destined for node 11.

destined for R (R is obtained from O_1 using the same hash function that S used). At some point (at latest when reaching R) the message will route through a node that has knowledge of O_1, which then is able to forward the message to S, as shown in Figure 4.6. If a replica of O_1 is published at another server S', the publishing works similarly, with the change that the message $\langle S', O_1 \rangle$ replaces $\langle S, O_1 \rangle$ at each node where S' is closer than S.

4.6 OceanStore

The OceanStore project [R+01b, K+00, REG+03] (http://oceanstore.cs.berkeley.edu) at the University of California at Berkeley is an attempt to construct a secure highly available and reliable storage system on a global scale. The system is envisaged to support the storage needs of some 10 billion users, amounting to a total capacity of roughly 10^{18} bytes. The distinguishing features of OceanStore are its massive scale, content-based routing, strong support for security, and use of introspective techniques to optimize the performance of the system. An open source prototype implementation of OceanStore, called “Pond”, was released in 2002.

The fundamental unit of storage in OceanStore is an encrypted binary object, identified by a fixed-length globally unique identifier (GUID). Objects are stored persistently and new versions are automatically created with each update. To give the illusion of mutable objects, a special GUID called the *active GUID* is used to access the latest version of an object.

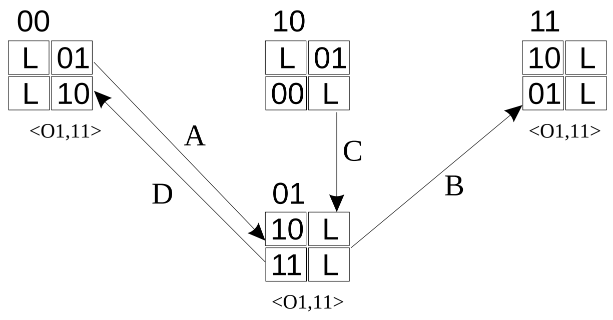

Figure 4.6: Object lookup using the algorithm of Plaxton et al. Node 00 publishes O_1, whose root node is 11. The arrows A and B indicate the path of the published message. When node 10 queries for O_1, it sends the query to node 11. However, the location of O_1 is discovered at node 01 after the first hop and the message is sent directly to node 00 (arrows C and D).

To facilitate fault tolerance and to increase performance, active objects are automatically replicated and distributed among network nodes as seen fit by the system. Older versions of an object are stored in a highly fault-tolerant *archival* mode. In archival mode, an object is encoded using erasure codes and divided into fragments that are spread over a large number of servers. The original object can be reconstructed from any sufficiently large subset (e.g. 30%) of the fragment set.

The mapping from object names (such as human-readable file names) to object GUIDs is accomplished by hashing over the object name and some additional information, such as an encryption key. Objects can also be named by hashing over the object's content, e.g. in the case of archived objects.

The devices participating in the OceanStore infrastructure are nodes in an overlay network[1] that implement KBR on top of an existing IP infrastructure. The main feature of the overlay network is its ability to route messages (e.g. data reads) to the "closest" instance of a stored object. The KBR mechanism is one of the fundamental enablers of fault tolerance and replication in OceanStore.

The architects of OceanStore have done their utmost to eliminate single points of failure from the system. This is reflected throughout the design and especially in the subsystem called the *inner ring* (or *primary tier*[2]),

[1]That is, an application-level virtual network on top of an existing physical one. A well-known example is Gnutella.

[2][K$^+$00] uses the term *primary tier*, later publications [R$^+$01b] use the term *inner ring.*

which handles global ordering and commitment of update operations, and provides a source of data with strong consistency guarantees. The task that the inner ring performs is typically handled by a single authoritative server in other distributed storage systems such as Bayou and Coda.

To the application developer OceanStore provides its services through sessions. Sessions may be created with different types of consistency guarantees like in Bayou. As expected, there is a tradeoff between consistency guarantees and connectivity: in cases of disconnection or weak connectivity, strong consistency cannot be guaranteed.

The basic assumption regarding security in OceanStore is that the infrastructure is untrusted. In practice this means that no node except the client is allowed to see unencrypted data, limiting read access to those in possession of the encryption key. Write access is controlled through signed access control lists. Measures have also been taken to prevent malicious servers from replacing or changing objects to which they do not have access rights.

OceanStore was designed to be self-maintaining. This implies two fundamental properties: fault tolerance (the mechanisms of which were described above) and self-repair. The mechanisms for self-repair in OceanStore include the ability to automatically handle both unexpected and advertised insertions and removals of nodes, processes that monitor the network for suboptimal routes and periodically sweep through OceanStore to check and repair objects (using archival fragments), as well as models for predicting server and disk failures.

To further reduce the need for manual tuning, *introspective* processes, i.e. processes that observe the system and tune based on these observations, have been deployed. Introspection is used for cluster recognition, whereby clusters of closely related files (e.g. a set of files a user is actively working on) are recognized. This clustering information, along with information on usage patterns, is utilized by another introspective process, the replica manager. The replica manager is able to intelligently prefetch files to a server close to the user: mail is automatically fetched to the user's workstation during office hours and to the user's PDA while traveling.

The use of fault-tolerant encodings, redundancy throughout the system, and introspective processes aid in achieving such desirable properties as scalability, resistance to denial-of-service attacks and censorship, and durability [Kub03].

4.6.1 Storage and Update Model

OceanStore uses a twofold approach for accessing data objects [K+00]. First, a fast probabilistic algorithm, based on Bloom filters, is used to look for the data in nearby nodes. If the probabilistic access fails, an approach based on KBR is used.

Bloom filters [Blo70] are a way to represent sets compactly with some probability for false matches. The nodes store Bloom filters for the objects in adjacent nodes up to some depth. If a filter indicates a match routing to the corresponding node can be done directly.

The KBR method is based on the Tapestry overlay and routing infrastructure (also developed at Berkeley) which is used for routing the messages to the closest instance of a stored object. Tapestry implements a variation of the KBR mechanism introduced by Plaxton, Rajaman, and Richa, with enhancements for fault tolerance and dynamic insertions and removals of nodes.

Each stored object is assigned a set of servers, which form the object's inner ring. The servers in the inner ring agree on updates using a Byzantine agreement protocol [LSP82], guaranteeing consistency among the replicas. The protocol allows any m out of $3m+1$ servers to fail without the ring going inoperational. The penalty for this high level of fault tolerance appears to be a relatively large amount of network traffic between the servers in the inner ring. However, in [K+00] it is argued that the overhead of the Byzantine agreement protocol is not in fact very large.

In addition to the replicas in the inner ring, there may exist *secondary replicas* of an object throughout the system, distributed in the form of trees rooted at servers in the inner ring. To summarize:

- A strongly consistent and current copy of an object can always be obtained from the servers in the object's inner ring.
- Objects are replicated. The replicas increase availability, but lack strong guarantees on currency and consistency.

Updates

An update to an object consists of a $(predicate, action)$ pair that is propagated through the overlay network. The action corresponding to the first predicate that evaluates to true is executed atomically, and the update is committed. If no true predicate is found, the update is aborted. Data locking is not used to avoid the problems traditionally associated with locks, e.g. stale locks and too aggressive locking preventing shared access.

These ideas are similar to those used in the Bayou system, with the update predicates acting as Bayou merge procedures, and the inner ring in the role of the Bayou primary replica. The fact that only ciphertext is stored in OceanStore limits the available predicates that can be used in the update operations, and the substitution of a primary replica with a group of servers makes the update procedure more challenging.

According to [K+00], updates may be propagated to the secondary replicas in three ways: the updates get propagated down the tree of replicas from the inner ring as shown in Figure 4.7, replicas may send requests for updates up the tree, or in an *epidemic* manner (as in Bayou). During an epidemic update, the replicas quickly spread tentative updates among themselves and pick a tentative serialization order. Tentatively updated replicas can be read by applications that do not require strong consistency guarantees. In [REG+03], which describes the current prototype implementation of OceanStore, only the first method of propagating updates is used: when the inner ring has agreed upon an update, it is multicast to all replicating instances.

The predicates available for updates are *block-compare*, predicates on metadata, such as *compare-size* and *compare-version*, as well as a *search* predicate for searching the ciphertext for a string (without revealing the cleartext of the search string). Operations such as *replace-block*, *delete-block*, and *insert-block* are also available if we assume that certain types of block ciphers are used.

ACID semantics can be achieved by using the *compare-block* predicate to check the read set of the transaction and then writing the updated data with *replace-block*. If an application requires reads to return data that is current and strongly consistent (e.g. a banking transaction) it needs to communicate with the inner ring, since the secondary replicas may contain outdated or tentative data. In theory, this means that a mobile device that requires strong consistency guarantees needs to be in the inner ring.

A question that the lockless transaction scheme raises is that of livelocking. Consider the transaction $\{a = a + 100, b = b - 100\}$ (which may signify the transfer of \$100 from account b to account a). With lockless transactions, the amounts a and b first need to be read, and then an update constructed that checks that the amounts are the same that were read before updating them. The concern here is that the read and update cannot be done atomically, so there is always the possibility that another transaction updates the amounts in between — in which case the update aborts and must be started all over. However, there is no guarantee that the update will succeed on the second try, or any subsequent one for that matter, i.e. the update may *livelock*. This issue has reportedly

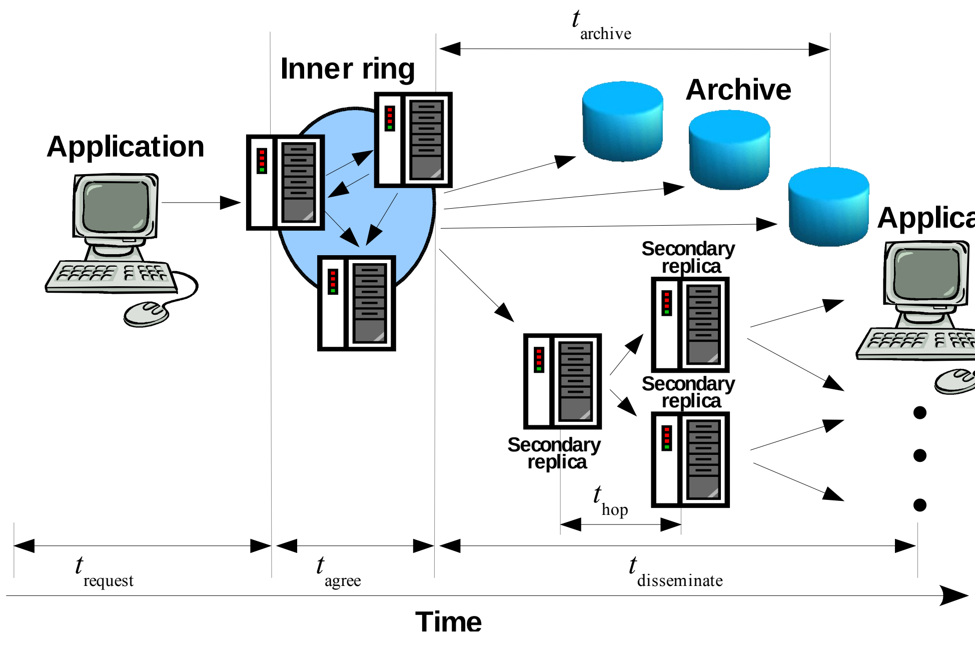

Figure 4.7: The path of an OceanStore update. An update proceeds from the client to the primary replica of its target data object. There it is serialized with other updates and applied to its target. The update is then multicast down the dissemination tree to other replicas. Simultaneously, the new version is erasure-coded and sent to archival storage servers [REG+03].

been encountered when implementing NFS [Now89] directory updates in the OceanStore prototype [REG+03].

The predicate-action update scheme is also able to handle simple conflict resolution automatically. As an example, if the most recent version should always persist, the *compare-version* predicate can be used to check if the version number has increased since the last read, and the write discarded if that is the case.

4.6.2 OceanStore as an MDIB

The authors of [K+00] mention the possibility of disconnected operation of OceanStore. The fundamental enablers are indeed there: local replicas can be stored on the mobile device and data writes need not propagate immediately. There are, however, several aspects of OceanStore which may not be optimal in the mobile environment:

Protocols There is no mention of the protocols used by OceanStore having been optimized to save bandwidth or round-trips. The amount of different protocols active in OceanStore appears quite large: client update requests, epidemically propagated updates, pushed updates to secondary replicas, update requests from replicas, several introspective processes exchanging system state, and processes sweeping through all the stored objects. In a mobile environment, presumably several of these processes would need to be disabled to conserve bandwidth.

Most importantly, data updates are currently not optimized for limited bandwidth. The use of ciphertext makes it harder to add incremental updates to the protocol, which is a common technique for minimizing bandwidth usage. This point is, however, not related to OceanStore in particular, but to any system storing ciphertext.

Complexity and Code Size The reviewed OceanStore prototype, which does not implement the full set of protocols described in [K+00] and [R+01b], already contains some 50,000 Java statements [REG+03], indicating that it may be too complex for current mobile devices.

Overhead of Strong Consistency Putting a member of the inner ring on a mobile device seems to incur a performance and bandwidth penalty, given the rather complex Byzantine agreement protocol. Implementation experiences from [REG+03] also show that the hosts in the inner ring are required to perform rather heavy computations for the purpose of archiving and cryptographically signing data.

Disconnections Unexpected disconnections are an issue that the system needs to deal with. Practical implications are that protocols should minimize the amount of state and that connection buildups and teardowns should be light. The update messages seem to be stateless and hence well suited for mobile operation. Connection buildups, on the other hand, may be problematic due to the initial traffic to a node joining a Tapestry network. The use of a KBR better suited for mobile usage than Tapestry might alleviate this problem, though.

As for security, the fact that OceanStore was designed with strong security in mind from the ground up is a merit in the wireless environment. Furthermore, OceanStore does not distinguish “servers” and “clients”, and should thus be able to work in peer-to-peer mode.

Although currently not ideal for deployment on actual mobile devices, OceanStore could very well play a role on the server side of an MDIB. Mobile access points could deploy OceanStore for synchronizing data among each other, whereas synchronization between a mobile device and its access point would be handled by a more lightweight approach.

4.6.3 Practical Issues

The source code for the OceanStore prototype is available on the web at `http://oceanstore.sourceforge.net`. Project activity during 2002 and 2003 seems to have been concentrated on development originating at Berkeley. During 2003, some 10% of the Java code in the CVS repository was changed[1]. OceanStore is licensed under a BSD-style license, making it easily available for experimentation and utilization in a research project. The underlying routing framework, Tapestry, is also available as a separate module at the OceanStore website `http://oceanstore.cs.berkeley.edu`.

To the programmer, OceanStore provides several alternative APIs. The base API provides full access to OceanStore functionality in terms of sessions and session guarantees, updates, and callbacks (which are used to inform the application e.g. when an update commits or aborts). On top of this API, more familiar APIs (called *facades*) may be implemented, allowing legacy applications to harness the benefits of OceanStore. As an example of this, the OceanStore prototype includes the ubiquitous Unix file system API [REG+03].

4.7 SyncML

SyncML (`http://www.openmobilealliance.org/syncml`) [Pab02, Syn02b, Syn02a] is an industry initiative to standardize the way data synchronization is handled on mobile devices. SyncML started out as an independent initiative, but joined the Open Mobile Alliance (OMA) in late 2002. The OMA Data Synchronization Working Group continues the work originated in the former SyncML Initiative. Participating companies include wireless heavy-weights Ericsson, Nokia, Motorola, and IBM.

Synchronization has traditionally been handled in an application-specific and often proprietary manner, which leads to limited interoperability between applications, lack of support for different transport methods, as well as an inconsistent user experience, due to differing designs. The goal

[1] `cvs diff` yields about 27,000 inserted and 10,000 deleted lines between March 2003 and February 2004

of the initiative is to be able to remedy this situation and enable “mobile devices that support synchronization with any networked data” (and vice versa).

To this end the SyncML initiative has specified a synchronization framework. The essential parts of the framework are the SyncML Synchronization protocol, the SyncML Representation Protocol, and bindings for various transport protocols such as HTTP [FGM+99] and OBEX [IRD03]. The initiative has also released a set of protocol specifications for management of device configurations [Syn01], the rationale being that the synchronization framework can be leveraged when synchronizing configuration data.

The *synchronization protocol* defines the interaction between a device and its peer. This entails connection setup, authentication, synchronization (in several modes), and object ID mapping procedures. The *representation protocol* presents the protocol messages in detail in terms of syntax, parameters, and result codes. The representation protocol also introduces the ability to filter and search the database as well as execute commands on the peer.

To illustrate the difference between these, we may for instance consider the authentication procedure: the former specification defines when and which messages are sent, while the latter gives the format for each of the messages sent. We will refer to both protocols collectively as the “SyncML protocol”.

The main features of the SyncML protocol are

- Multiple transport protocols. SyncML can work over HTTP, OBEX, WSP, and Bluetooth.
- Support for synchronization of any data that can be expressed as a collection of (*key,value*) pairs. Values may be arbitrary binary data, including XML documents, vCards, email messages, etc.
- Optimization for the mobile environment.[1]
- Use of existing technologies such as XML, HTTP, and TCP/IP.

The SyncML architecture is client/server, the mobile device normally being the client and its strongly connected peer the server. The synchronization mechanism is based on transmission of updates between the client and the server. Typically, the client sends its updates to the server, which reintegrates them (possibly solving conflicts). The server then sends

[1]The author has been unable to locate any further references on how this optimization was done.

its set of modifications (including possibly resolved conflict entries) back to the client, which stores them in its database. The update operations allow insertion, deletion, replacement, and copying of objects.[1] Several modes of synchronization are supported, used to bring either or both devices up to date, and to account for the possibility of a missing update log:

Two-way synchronization Client-initiated, updates are transferred from the client to the server and vice versa.

Slow sync The client transmits its entire dataset to the server, after which the server transmits updates to the client. Used instead of two-way synchronization if the client update log cannot be used for some reason (e.g. the log is lost).

One-way sync from the client Updates are transferred from the client to the server only.

One-way sync from server Updates are transferred from the server to the client only.

Refresh sync from client The client sends its entire dataset to the server, overwriting any corresponding data in the server database.

Refresh sync from server Like refresh sync from client, except that the entire server dataset is transferred to the client.

The SyncML protocol does not specify how conflicts are resolved; this matter is left to the implementation of the synchronization engine. The protocol does, however, define some messages and status codes relating to conflict resolution, e.g. a status code indicating that a conflict was resolved by merging conflicting records.

To know from which point in the update logs synchronization should occur, as well as to enable devices to synchronize with multiple peers, synchronization *anchors* are used. The anchors mark positions in the update logs (similarly to how positions in the KML in InterMezzo are remembered) for each peer, and are used to check that no updates are applied more than once or lost on successive synchronizations.

When adding items in disconnected mode, there is the question of how to allocate new object IDs. In systems such as Coda, this is solved by tentatively assigning IDs, which are then verified upon synchronization. SyncML takes a different approach: each client manages its own set of

[1]From a theoretical standpoint, it is interesting to note that there is a `copy` operation but no `rename` operation.

IDs (called local IDs). During synchronization a map (stored on the server) between local IDs and server (global) IDs is generated.

The authentication methods supported by SyncML work similarly to HTTP basic and digest authentication.

4.7.1 Using SyncML in an MDIB

Since the synchronization problem SyncML sets out to solve and standardize has significant overlap (data synchronization over the wireless hop) with that considered in the Fuego Core project, we should carefully consider using SyncML technology when applicable.

Therefore we evaluate SyncML by analyzing it with respect to the definition of synchronization used in this report, i.e. with respect to update detection, update propagation, and conflict resolution. In the analysis we have included the SyncML protocol, as well as two publicly available implementations: the SyncML C toolkit and the `sync4j` Java implementation.

Update detection The SyncML protocol does not specify how to determine which data items have changed since a previous synchronization. Update detection is thus not addressed by the specification. The C toolkit also lacks this functionality. The `sync4j` implementation, on the other hand, includes some concrete implementations (e.g. the synchronization of files in a single directory).

Update propagation The SyncML protocol defines update propagation, and both the C toolkit and `sync4j` provide concrete implementations.

Reconciliation Reconciliation is addressed by `sync4j`, which provides concrete implementation examples. The SyncML protocol and the C toolkit do not address this issue.

The SyncML protocol by itself or in combination with the C toolkit can thus be used to provide a means of propagating updates between devices, the benefit of the C toolkit being that we would not have to implement the generation and parsing of SyncML messages. Using SyncML for update propagation would enable communication with other SyncML-enabled devices, as well as make it possible to use any of the transports supported by SyncML.

However, if we want to optimize bandwidth usage by employing more efficient encodings (including delta transfers), we would have to implement our own SyncML transport. Furthermore, adding any type of “intelligence” to the synchronization process will most likely require customized code on both ends of the connection. Given this scenario, it seems easier to bypass SyncML entirely, and just transmit updates using a simple binary transfer protocol such as HTTP. At the same time we would be able to bypass limitations such as the SyncML data model, the need to encapsulate BLOBs inside XML, and the need to solve conflicts on the server.

The `sync4j` implementation would potentially be a better starting point, since it looks promising for experimenting on synchronization policies, e.g. when and what to synchronize. There is, however, the issue of complexity: the SyncML specification appears quite complex, both to use [Fan02, Buc02] and in terms of implementation size. Indicative of this is that the C toolkit is some 54,000 lines and `sync4j` some 56,000 lines.[1]

4.7.2 Deployment Issues

A Java implementation of the SyncML framework, a SyncML server, and practical examples are developed in the Sync4j (`http://sourceforge.net/projects/sync4j/`) project. The project provides a comprehensive implementation of SyncML-based synchronization. Sync4j is a valuable aid to developers learning to utilize the collection of SyncML specifications. The code is available under the BSD license with some reservations regarding patented parts of the protocol.

4.8 Synchronization Policies

It is generally possible to associate a set of parameters with the data synchronization process. Examples of such parameters are

- *when* to synchronize (at the user's request, at regular intervals, etc.)
- *what* to synchronize (synchronize everything, synchronize important data first, etc.)
- *transport method*, i.e. choosing appropriate encodings (e.g. fast versus bandwidth-conserving) and interfaces (e.g. GPRS versus Bluetooth).
- *reconciliation method*, i.e. how should conflicts be resolved.

[1]The line counts were obtained by counting `.c`, `.h` and `.java` files in the SourceForge CVS repositories on Feb 18, 2004

We define a *synchronization policy* to be a set of parameter values for the synchronization process. For instance, the `mobile` policy may be set up as `{when=ask, transport=GPRS}`, whereas the `home` policy is `{when=hourly, transport=ethernet}`.

To the author's knowledge, there seems to be no work on synchronization policies in general. This is quite understandable, as each particular synchronization system has its own set of assumptions and parameters for tuning the process. Furthermore, the boundary between what constitutes a synchronization algorithm and what constitutes a parameter is quite fuzzy.

Some examples of policies were obtained by studying existing systems:

What In [MES95] a model of user patience is used to determine whether a cache miss should be handled transparently or the user should be asked for confirmation before fetching the object. The threshold depends on available bandwidth and the priority of the object (which the user can set). For instance, a 1 megabyte object at medium priority would be fetched transparently at a link speed of 64kbps, whereas the user would be asked for confirmation on a 9.6kbps link.

[HKZ02] describes an extension to Coda that allows users to either specify the maximum amount of time or money that should be spent on reintegration. The implementations of the policies are rather straightforward: they simply stop the reintegration process whenever the maximum has been reached — there is, for instance, no prioritization in what objects to fetch.

Transport Method [Mob01] describes an "intelligent delta selection process" which is employed in a commercial synchronization tool called Network/Unplugged. The tool utilizes deltas for object synchronization, and different types of deltas can be used depending on user needs. The available methods are block-level differencing, byte-level differencing, and write-monitor differencing. Detailed descriptions of these methods are not publically available, but from [Mob01] it appears that byte-level differencing essentially works as Unix `diff` and block-level differencing as `diff`, but where each token corresponds to a block of bytes. Write monitoring works by gathering a modification log, which is passed to the peer.

Each of these methods presents a different level of tradeoff between bandwidth and computing resources. Block-level differencing uses the most amount bandwidth, but requires the least amount of CPU cycles. For certain files (such as databases), write monitoring uses the least amount of bandwidth, but the most amount of CPU cycles.

When The original implementation of Coda [SK92], which did not have support for weakly connected mode, exhibits a very simple policy of when to synchronize: synchronize when the device is reconnected to the network. The same type of simple policy is employed in Microsoft's IntelliMirror [Mic99a] for network folders made available for off-line use.

Reconciliation Method The way InterMezzo solves conflicts depends on the active policy. The available conflict resolution methods are “Mobile”, “High Availability”, and “Re-synchronization” [Bra02].

4.9 Generic Data Reconciliation Methods

An important part of the synchronization process, especially when no strict concurrency control is employed, is the data reconciliation phase. In the Fuego Core project we are particularly interested in investigating *generalized* methods for data reconciliation, as opposed to application-specific data reconciliation.

The ability to perform automatic reconciliation is particularly important in the mobile environment, as many applications become “involuntarily collaborative”. Consider a simple text editor for instance: in the desktop world, authors may take turns editing a document. In the mobile world, on the other hand, one of the authors may be disconnected from the network for an extended amount of time, making the approach of taking turns infeasible and raising the need for disconnected collaboration.

A solution is to add data reconciliation capabilities separately to each application. Unfortunately, this will complicate application development, making it less attractive to write mobile applications. Applications will also be larger, wasting the already constrained resources on the mobile device. There is thus a need for data reconciliation facilities that incur little overhead on application development and on the size of the executable.

An important aspect of a reconciliation algorithm is the data structure it is designed to operate upon. These include sets of tuples (for relational databases), ordered lists (for text files), trees and graphs (for structured data), as well as application-specific structures. As data structures become more complex, the number of ways of changing the structures and integrating the changes increases, adding to the complexity of the merging algorithm. In addition, the semantics of the data structures may affect how the merging should be performed.

Despite these difficulties, general reconciliation algorithms for complex data structures are clearly useful. Frequently, an understanding of the full semantics of the data is not required for successful reconciliation. For instance, consider reconciliation by three-way merging of source code using the Unix `diff3` tool: the semantic structure of the source code may be very complex, yet valid results are often obtained from `diff3`, which considers the structure of its input data to be an ordered list of text lines.

The most common approach is to not include any generic reconciliation capabilities at all, i.e. to treat data objects as atomic units whose content is totally opaque. Instead, hooks for data reconcilers are provided. The reconcilers are typically called when the system detects concurrent modification. Coda and Bayou both take this approach[1].

The Concurrent Versions System (CVS) (`http://www.cvshome.org/`) supports two types of files: binary and text. Arbitrary binary files can naturally not be automatically reconciled, but on text files, CVS uses a `diff3`-like algorithm to reconcile changes. Any conflicts encountered during reconciliation must be resolved outside CVS (e.g. by the user). CVS is perhaps the most well-known system with generic (although limited) reconciliation capabilities.

Moving towards more complex structures, we arrive at the *tree*. It is a particularly interesting data structure to reconcile, as trees are general data structures expressive enough to accurately present the structure of the vast majority of application data. The use of a generic reconciliation engine for tree-structured data encoded as XML is discussed in [Lin03].

More complex structures (e.g. graphs) than the tree seem to quickly become intractable, especially in the general case. As an example, consider the program merging algorithm presented in [HPR89].

4.10 Strategies for Conserving Bandwidth

Given the often limited capacity of wireless links we need to optimize the amount of data transmitted. In this section we discuss strategies for conserving bandwidth that are applicable to the update propagation phase. The optimization of the set of changes obtained during the update detection phase is not considered.

[1]Note that the Bayou merge procedures are supplied by the application

From information theory we know that the entropy or “information content” of a message is always calculated with respect to some model. The more probable (“unsurprising”) the message is with respect to the chosen model, the fewer bits are required to encode it.

The bandwidth conservation strategies we mention here are

- Data compression,
- Delta transfers,
- Content adaptation, and
- Operation shipping.

With some generalization[1], these strategies represent increasingly expressive models to encode data, yielding the potential for increasingly efficient encodings. The models are, however, also increasingly specialized, meaning that the set of data that may be efficiently encoded becomes more and more restricted.

4.10.1 Data Compression

Compressors for the transport of arbitrary byte streams are already widely used for minimizing bandwidth usage on many levels in the protocol stack (e.g. Modem line encoding, IP header compression in the Point-to-Point Protocol, and `gzip` [GA02] content encoding in HTTP replies).

The use of generic compression in synchronization appears quite beneficial, as it should provide significant savings in several cases (e.g. text files and uncompressed graphics). However, as stated earlier, one needs to take into account the computation resources required by the algorithm. Another issue is the integration of encryption and compression, as encrypted data will not compress.

4.10.2 Delta Transfers

Delta transfer, i.e. the transfer of changes, can be employed when a previous version of the data is available. As the previous version frequently is quite similar to the new version, the delta may be significantly smaller than the original data.

[1]E.g. generic delta transfers may be less specialized than XML-specific compression

The systems discussed make use of delta transfers on some level: Coda only transmits modified files upon reconnection (instead of blindly copying the entire file tree), as does InterMezzo. Bayou servers send write logs instead of the entire database when synchronizing with their peers. Delta transfers can, however, potentially be used to a much higher degree in these systems, by utilizing delta transfers for individual files (binary objects) as well.

In [HKZ02] a modification to Coda is presented where file updates are handled by delta transfers. The deltas between files are generated by running the well-known Unix `diff` utility on the old and new versions, the old version being known to exist at the receiving end. The output of `diff` is transmitted over the network, where it is used to generate the new version. Experiments on text files in [HKZ02] indicate 65–95% savings in the number of bytes transferred between client and server. Although these numbers are overly optimistic (no transfers of binary files were included, since the standard `diff` cannot handle these) they show the potential of delta transfers on the file level.

A file synchronization algorithm explicitly developed for high-latency low-bandwidth links is presented in [Tri99]. The algorithm, implemented in the freely available `rsync` tool, can be used on both binary and text files, and does not assume that a particular version of the target file exists or is retrievable on the receiving end, as is the case in [HKZ02]. However, the more similar the existing version on the receiving end is, the more bandwidth can be saved.

Assume that we have two devices A and B, and we want to transmit a file F from A to B. On B there exists a file F', which is somehow related to F (e.g. F is a newer version of F'). The basic[1] rsync algorithm consists of three stages:

1. B splits F' into blocks and calculates a message digest[2] for each block. Let the digests be $S_1 \ldots S_i$. The digests are transmitted to A.

2. On A, F is scanned for blocks whose digest match one of the digests $S_1 \ldots S_i$. As a result, F can be expressed as a sequence of verbatim bytes from F interleaved with references to matching block digests. This sequence is transmitted to B.

[1]For brevity, several important details have been omitted here, such as the use of two types of message digests.

[2]The message digest is much shorter than the block. A block may be e.g. 1 KB while the digest is 16 bytes (the size of the MD4 digest originally used).

3. B now constructs F using F' and the received sequence. When verbatim bytes are read from the sequence, these are written directly to F. When a reference to a matching digest is read, the corresponding block from F' is written to F.

Thus, if for instance both files are binary files, and F would be identical to $b_1 \ldots b_k F'$, where $b_1 \ldots b_k$ are some bytes inserted at the start of F', the algorithm would transfer the sequence $b_1 \ldots b_k$ followed by references to the digests $S_1 \ldots S_i$. Assuming a block size of 1024 bytes, 4 bytes to index the digests, $k = 1024$, and F' approximately 1 MB in size, roughly 5 KB of data would be transmitted instead of 1 MB. In contrast to `diff`, the algorithm is also capable of handling moves of data (which show up as block reorderings).

In practice `rsync` may perform much worse due to small but scattered changes (block matches are destroyed by a few non-matching bytes), as well as compressed data (changes in data at position i may affect all bytes emitted by a compressor after it has passed over position i). Some interesting thoughts on how to remedy this problem are given in [Tri99].

4.10.3 Content Adaptation

In some cases it is possible to reduce bandwidth consumption by modifying the data itself. Examples of content adaptation include changing the data format, color removal and resampling of images, reducing the frame rate of video streams, and filtering of newsfeeds according to user preferences.

As content adaptation and transcoding is a large research field of its own, we point the interested reader to other publications, such as [JHE99]. Especially the WWW is an application that lends itself quite easily to content adaptation (e.g. by restructuring the HTML code and modifying images). For an example, see [LHKR96].

4.10.4 Operation Shipping

In operation shipping [LLS02] the client ships the user operation that updated the data, rather than the data itself, across the network. A user operation may be, for instance, a program invocation (e.g. `cc myfile.c`), or an operation logged from interactive use of an application (e.g. rotating an image).

As an example, consider a binary program file `hello`, which is generated by compiling the C source code with the command `cc hello.c -o` `hello`. The idea of operation shipping is to transmit the above command instead of the `hello` binary over the network.

Operation shipping may offer significant bandwidth savings: the experiments conducted in [LLS02] yielded 12- to 400-fold improvements. The drawbacks are that the operations and their relation to the data objects need to be determined somehow, as well as the need for a shared configuration (e.g. a C compiler) on both ends of the connection.

4.11 Conclusions

Among the reviewed systems there is none that would readily fit the role of a mobile distributed information base. Each system has its strengths and weaknesses. To design an information base for operation in a mobile environment we need to combine the best aspects of these systems. Coda and InterMezzo appear to be the most promising starting points. In Coda's case there is a certain maturity emerging from over a decade of research, the fact that it addresses several of the concerns of the mobile environment, and freely available source code. InterMezzo, being derived from Coda, should have many of its strengths, as well as a simpler design, and be well suited for mobile devices.

OceanStore is an interesting concept for implementing massive scalability, fault tolerance, and high availability on the server side. However, it appears that OceanStore is a bit too complex for current mobile devices. Furthermore, OceanStore itself is a very bold research project that explores a multitude of new solutions, and hence relying solely on OceanStore for our research appears to be taking quite a risk. A less bold approach in Fuego Core would be to design with OceanStore in mind for the fixed infrastructure, but allow for other storage solutions as well.

Bayou is especially interesting in the sense that it is the only system working as a shared database (as opposed to a shared file system). If it appears more useful to develop a database type of MDIB, Bayou is the natural starting point. However, the source code for Bayou is not, as far as we know, publicly available, requiring considerable amount of programming to even get a base system up and running.

It should be noticed that there are several design features that all of the systems have in common. If we choose to design an MDIB on our own, we should carefully consider before deviating from these:

- Acceptance of the tradeoff between strong connectivity and consistency versus weak connectivity and consistency. There is an agreement that strong consistency on weak connections is infeasible.

- Use of aggressive caching. Indeed, it is hard to imagine providing high availability in disconnected mode without aggressive caching.
- Support for write access while disconnected. Read-only operation during disconnection is not a viable alternative.
- The use of optimistic replication, with subsequent reintegration and conflict resolution. Write or read locking is not used in any of the systems to avoid conflicts.

Some issues in this review appear to be orthogonal, and it should thus be possible to "pick and choose" the best approaches from different systems. As an example of this, consider e.g. cache coherency approaches and object transfer mechanisms: it should be possible to modify the object transfer mechanism (by using such techniques as delta transfers) without affecting the cache coherency mechanism.

An important question, which this review touches only on the surface, is that of using XML as a storage format, for which we can provide enhanced functionality. Especially the possibility for generic reconciliation support appears interesting.

We need to address the question of whether to base the information base design on a database or a file system. These approaches have different optimal usage scenarios. Databases are more suited for retrieval and storage of single records in a vast dataset whereas file systems are more efficient and easier to use for traditional "document-based" applications, such as text editors, drawing tools, etc. An interesting approach would be to combine both, e.g. by allowing the file system to be aware of the structure of files containing XML data.

At least two possible paths of continued research can be envisioned:

1. Developing the underlying synchronizing file system or database, but not yet address such issues as synchronization policies.
2. Concentrate on developing policies. A synchronization policy engine would be constructed on top of an existing prototype information base, such as e.g. a combination of Coda and SyncML.

Naturally, combinations of these are also possible. For instance, it would seem to make sense to pursue the first alternative as far as to have a prototyping platform, after which one could try approaching the issue of policies.

Chapter 5

Mobile Presence

5.1 Introduction

This chapter reviews the current state of presence services, with special focus on the needs of mobile computing. Presence services are here understood as application features or components that deliver elementary information about the current state of a person to a select group of peers. The information typically consists of hints about the current availability of that person for synchronous communication, such as a phone call. Other types of possible information include, for example, the current location of that person.

Presence services are often integrated to group collaboration or other communication tools, most notably in instant messaging systems. Existing instant messaging technologies are not interoperable on protocol level but some specify server-to-server protocols, enabling gateways between technologies. Interoperability work in the IETF Instant Messaging and Presence Protocol (IMPP) working group is being considered by the bodies specifying open Instant messaging and presence (IMP) technologies.

5.1.1 Concepts

By definition, presence information describes the current state of a person, in the sense of present versus absent: “Mike is at the office.” The information can also be more fine-grained, like contact hints: “Steff is at the office, in a meeting. He can be paged but not called.” Presence services are sometimes referred to as “group awareness software”.

Since instant messaging is a very convenient application for presence information, it is often coupled with it in various implementations and technologies. These two types of technologies are commonly referred to as IMP in this chapter.

5.1.2 Presence and Context

The concept of a presence service is easily confused with the concept of context awareness. Although they share many similarities, there is a distinction to be made. Context awareness in computing usually refers to intelligent software that adapts to the situation, a context, that a user or his devices are in. The term “Context awareness” is used to reference many different technologies, from understanding and reacting to the social context of the user to adapting to the network link layer context.

Presence information is more like a collection of singular pieces of information that may be interrelated. Some of this may be sensor (noise level) or application level (phone up/down) data. The interpretation of this is typically left to the user. In the most basic form presence information describes the availability of a user for interruptive communication. [DA99]

5.1.3 History

In the early 1980s there were a number of different locally-used methods and occasional prototypes for finding out if a particular person is currently logged into a network or a mainframe, and for sending them messages. Early such tools include

write a method of sending/writing text to the terminal of another user

talk an interactive program that enables two persons to chat on the same machine

who lists users currently logged in on the local machine

finger a method for remotely probing information about a particular user or users

Internet Relay Chat

The earliest popular instant messaging system that supported a limited kind of presence information was the Internet Relay Chat (IRC) [OR93]. IRC was first implemented in the year 1988 and rapidly gained wide adoption, particularly among computer hobbyists. An IRC service consists of a network of servers that relay messages and user information between one another. Each server may accept any number of connected clients to which it serves messages and user information from the network.

For messaging IRC currently supports both group chats and private one-to-one messaging, as well as content sharing. Group chat participants are able, to certain extent, set policies and access control rules enforceable by the IRC servers. Content sharing is achieved by P2P transfers initiated by client-to-client messages through the IRC service. For presence information an IRC network provides

- the network address a user is logged in from,
- user-settable away status and an optional explanatory message,
- the time since a user has last said anything (activity), and
- the local time of a user

IRC networks, despite their design problems, continue to thrive as a global instant messaging and presence service, presumably owing much to the open and standard protocol. Innovative IRC clients range from full-fledged desktop applications such as mIRC (`http://www.mirc.com`) to the very minimal Short Message Service (SMS) or WAP interfaces [Jus01]. The openness of the protocol has sprouted many service extensions and new features overlooked or not specified in the original RFC:

formatting A non-standard extension to embed color coding and some other formatting instructions to message text

character sets The need for internationalization (i18n) has manifested as conventions for encoding various character sets (Unicode, Japanese, Cyrillic)

user accounts Some IRC services provide long-lived accounts, permanent chat room ownerships, and user authentication.

The IMP system Secure Internet Live Conferencing (SILC) [Rii03a], commonly comprehended as "secure IRC", is discussed on page 132. It is strongly influenced by IRC and the design addresses many problems identified in the IRC protocol and architecture.

ICQ (I seek you)

In 1996 an Israel-based company Mirabilis released its new IMP product called ICQ. ICQ service architecture was based on the client-server model where the servers were all provided by Mirabilis, although clients could send each other instant messages and data directly when possible. The user client had a simple GUI that showed the status of friends with icons, and much of its behavior and look is imitated in later IMP products. ICQ gained user adoption quickly and, by publishing their proprietary protocol specification, Mirabilis encouraged the creation of third-party clients. This helped the adoption of ICQ on many different platforms. ICQ will be discussed also on page 133. The Mirabilis ICQ client managed to popularize the now-common instant-messaging-related concepts of intrusiveness, network security, and privacy concerns.

On the wake of ICQ's success other competing IMP services soon became available. Companies like Yahoo! (`http://www.yahoo.com`), Microsoft (`http://www.microsoft.com`) and America Online (`http://www.aol.com`) were all highly successful in establishing global and popular IMP services. The wide adoption of these highly usable and instantly available services is now also being seen in the workplace [IWW+02], which raises new issues about privacy, confidentiality, and security.

5.1.4 Challenges

In recent years IMP applications have become commonplace on the desktop, and it is expected that mobile appliances are next. While many issues remain unsolved on the desktop world, the mobile world presents a whole new problem set, but perhaps also more mature and standard technology.

Interoperability

Despite the great success and adoption rate of IMP services there is so far no widely adopted standard protocol or global interoperability between the established networks. Many highly popular IMP services use a bundle of a proprietary client, server, and protocol. A client from one IMP service provider is unlikely to be interoperable with any other service. Also, messages and presence information are not exchanged between service networks. If a user wants to communicate with a peer on another service provider, he must also subscribe to that other service. Although there are technical solutions where the client software is able to communicate simultaneously with several different networks, the user must still subscribe and maintain his information in all of them.

If IMP is ever to achieve global success similar to electronic mail, service provider interoperability is paramount — otherwise IMP may remain a service provider value-added service or a niche application. In small mobile devices the embedded software may be hard to update and interoperability by supporting a multitude of different technologies is not feasible.

Security

There are many aspects of security that are of interest in IMP services and applications. Firstly there is privacy: a user must at all times be able to control what information is provided of him and to whom. Most solutions accomplish this by empowering the user with reciprocal buddy lists, promising that the server will honor that. Another question is how good that promise is when bad business practices, lacking security, or local legislation may void it all. This is also related to the problem of verifying the identity of a peer: when talking face-to-face, identifying a friend is much easier than when talking over a network. The richer and more ubiquitous presence services become with mobile devices, the more crucial proper management of access control is [HS96].

Another privacy issue, especially prominent in instant messaging, is unwelcome communication. If a user is unable to restrict who may send him a message and when — or in the case of mere presence, a subscription request — he may become a victim of a flood of unsolicited messages and interruptions, similar but perhaps even worse than the current situation with electronic mail.

Confidentiality is an issue that is typically secured on a hop-to-hop basis by encrypting the network traffic. This still has issues for organizations that are unwilling to trust a service provider for the security of their internal or trusted client communication. Some technologies support end-to-end encryption, yet even the knowledge of who are communicating (or are buddies) may be considered confidential: a service provider in an architecture where the server stores and manages contact lists for each client has a very comprehensive database of personal contact networks. This problem can be partly overcome either by setting up in-house organizational IMP networks, or by having the system support confidential peer-to-peer communication.

A requirement that contradicts privacy has recently been raised regarding the instant messaging systems. In some countries public companies are required, by law or due to other interest, to keep track of all their written or electronic internal communication. The goal here is to preserve a record of the internal communication so that it may be referred to later, perhaps to audit the legality of business practices. To this end all internal instant messaging traffic would have to be intercepted and stored. With increasingly heterogeneous systems and mobile communications this may prove technically challenging.

Mobility

The main issues with mobility are common to any technology applied to a mobile environment: highly limited resources, multiple access points and clients, unreliable communication, and ubiquity.

Recent developments in mobile devices have greatly pushed back the limits of programmability and software complexity. While these devices are still much more limited than desktop computers, they now allow deployment of quite advanced applications from intelligent agents to video games. Presence applications considered, modern devices are sufficiently equipped to be an active part of a presence service, while user interface limitations and reliance on battery power remain restrictive factors. Having an application that frequently sends and receives data updates over the air is a constant drain on battery power. Also, directly interacting with systems designed primarily for desktop usage may result in excessive traffic amounts, since acceptable bandwidth usage is typically on a whole different scale. Solutions like update collation, server-side filters, and context awareness are some possible remedies for unnecessary wireless traffic.

The trend where small mobile devices become network-aware and programmable creates new challenges for presence applications. Making several devices provide and consume presence data enables refining it to be more informative and accurate. Instead of a single device providing a limited view of the presence of a user, this view can be collated from multiple sources. This presents the problem of managing a frequently changing set of different client applications. The devices, ranging from a home PC to a pager, may differ greatly in software capabilities and connectivity, yet they should be able to collaborate dynamically.

Although ubiquity is usually thought of as desirable, it may well have adverse effects in the case of presence services. A presence service user in an ubiquitous environment must be aware of what kind of data about his presence is constantly being broadcast into the network. The setting should be such that a user is able to easily assert the limits of this data.

5.2 Existing Presence Technology

There are several factions driving standard enabling technologies for IMP. Most are consortiums driven by the software and telecommunication industries. This section provides a brief introduction to some key public players developing IMP technology. Some existing technologies and products are also reviewed briefly.

5.2.1 Open Mobile Alliance

OMA (`http://www.openmobilealliance.org`) is an industry consortium dedicated to developing common and open technologies for enabling development of mobile services. OMA has assimilated such consortiums as WAP Forum, Wireless Village (WV), and Location Interoperability Forum.

Wireless Village

Before merging into OMA the WV Initiative released two versions [WV02a, WV02b] of a comprehensive IMP system, titled The Wireless Village Specification. This specification covers everything needed from access control, server-to-server communication, and content sharing to different transport bindings.

The WV architecture is a strict client-server model. It specifies two client-server protocols, a full-feature XML protocol for networked applications and a limited-feature Command Line Protocol for more constrained environments. The XML-based protocol uses a query-response model, where each sent message package is a stand-alone XML document. Although the WV specification was created largely by vendors in the field of mobile communication, the XML protocol has been criticized for its average message size, which is large enough to cause noticeable delay in service interaction.

The WV specification covers the interfaces between components — protocols, functions, and data types — in detail, with little ambiguity. They cover presence handling, messaging, contact list management, security, and file sharing. This is important since components from different vendors should be readily interoperable. The WV technology has been and is being implemented by several vendors.

The integration of WV work to OMA has been completed and split to proper working groups and committees. The WV specification is being replaced by the more recent OMA IMPS version 1.2 specification set

[OMA03].

Location Interoperability Forum

Like Wireless Village, Location Interoperability Forum (LIF) started as a standardization effort and was later merged into OMA activity. The working group strives to produce end-to-end specifications for mobile location service interoperability. The Location working group (WG) has not been chartered to specifically coordinate its work with either Messaging or Presence & Availability WGs. The architecture is to cover the primary aspects of location services, application and contents interfaces, privacy and security, charging and billing, and roaming. Since location information is a classical example of primary mobile presence information, it will be interesting to see how well these two evolve to interoperate within OMA.

5.2.2 The Parlay Group

> The Parlay Group is a multi-vendor consortium formed to develop open, technology-independent application programming interfaces (APIs) that enable the development of applications that operate across multiple, networking-platform environments.[1]

The Parlay Group (`http://www.parlay.org`) is a vendor consortium and the APIs they develop are for facilitating the engineering of mobile software and integration of different systems and for driving innovation by reducing the cost of developing applications. The work is intended to benefit operators, service providers, and software vendors.

Presence and Availability Management Forum

The Presence and Availability Management Forum[2] was a consortium of companies aiming to define common interfaces for flexible management of private presence related data. This covers the definition and handling of identity, access control, and availability. The motivation is both to put the user in control of his personal information and to provide a common way for application developers of finding out, negotiating, and applying user privacy preferences. The Presence and Availability Management (PAM)

[1]`http://www.parlay.org/about/`
[2]`http://www.parlay.org/about/pam/index.asp`

specification covers the architecture definition and a description of a general level software API, without any language bindings.

The PAM Forum work was transferred to the Parlay Group PAM Working Group during the year 2003. The work to address the important issues of identity, access control, and control domains continues therein.

5.2.3 Internet Engineering Task Force

The Internet Engineering Task Force is closing in on IMP from several angles. This chapter discusses in detail only the Extensible Messaging and Presence Protocol (XMPP), SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE), and SILC technologies.

The IMPP working group has produced requirements and a reference architecture design for IMP systems, as well as a common data format for gatewaying between different systems [DAMV00, DRS00]. Several other working groups (Application Exchange (APEX), SIMPLE, and Presence and Instant Messaging Protocol (PRIM)) were charged with defining IMPP-compliant concrete end-to-end protocols in collaboration with the IMPP working group. Of these the APEX group produced a single draft, which was later accepted as an informative RFC. The PRIM disbanded due to inactivity without producing anything, and SIMPLE is close to producing a set of IMPP-related RFCs for IMP.

Extensible Messaging and Presence Protocol (XMPP) and Jabber

The XMPP work is based on the existing and widely adopted Jabber protocol. The forthcoming XMPP RFC will address the problems, oddities, and missing critical features in the original Jabber protocol, for example in the fields of security, interoperability, and internationalization. In essence evolving Jabber into a sound standard.

The Jabber protocol is an open and free protocol that is governed and directed by the Jabber Software Foundation (`http://www.jabber.org`). The Jabber protocol is generally considered moderately compact, easy to implement, and very extensible. The core protocol is quite simple and presents few requirements on network services (only the availability of TCP/IP) and no requirements on middleware. This simplicity is attractive to new implementers and there is an abundance of different server and client implementations. Client implementations are available even for mobile (Connected Limited Device Configuration (CLDC)) devices.

The Jabber protocol is essentially an asynchronous bidirectional stream of two continuous XML documents, enclosing messages of simple XML-formatted predefined types. The XML format provides three major advantages:

- human readable protocol for development and debugging,
- ability to use existing XML libraries for parsing and generating the protocol content, and
- virtually unlimited and safe message structure extensions through standard XML extension mechanisms.

The way XML is used in the Jabber protocol is unconventional and the corrections — in relation to both XML and other issues — that the XMPP working group needs to make break backwards compatibility. This compatibility issue is noted in the XMPP WG charter and the Jabber Software Foundation (JSF) has declared that it is committed to support the necessary modifications. The working group has submitted the core protocol specification [SA04b] for RFC approval and is continuing on more detailed work, mainly considering security and interoperability [SA04d, SA04a, SA04c].

The Jabber deployment architecture is a client-server model, in which internetworked servers represent and manage user identities in their own particular domains. User accounts are unique in a server domain and, combined with the domain identities, supposedly globally unique. Jabber supports resource specifiers for client identities; these are names that can denote a specific service endpoint associated with a user identity. For example "joe@bar.com*/register*" and "joe@bar.com*/pda*" refer to the same identity but possibly different endpoints.

Although message traffic is not peer-to-peer, the Jabber developer community has defined extensions by which two clients may negotiate the establishment of a direct connection for arbitrary use like file transfer.

The XMPP/Jabber architecture and protocol choice has the advantages:

- service network growth through open protocols and server domain association through ordinary domain name lookups,
- clients can operate while behind a firewall and/or network address translation (NAT), and
- domain server has control over the accounts in that domain, making it easier to secure an organizational access point.

The fact that the Jabber protocol is a relatively compact XML protocol makes it attractive for mobile use. Problems may arise if the mobile platform does not support TCP connections. Also, the protocol poses no limitations on the maximum size of a message: the originating end with a broadband connection may pack along nice-to-have information that needlessly increases the message size. In both cases, a special mobile-aware server and client might be able to provide ways for more effective communication and perhaps message content shedding.

SIP for Instant Messaging and Presence Leveraging Extensions

Session Initiation Protocol (SIP) [RSC+02] is a specification of a middleware service for locating mobile and transient service endpoints and for negotiating the establishment of various connected sessions like phone calls or video transmissions. SIP has already been widely adopted — both software and hardware products exist — in communication applications, for example in telephone systems with built-in support for conference calls and digital services. The IMPP-compliant SIP extensions needed for IMP are being drafted to a RFC in the SIMPLE work group.

SIP provides services for endpoint locating, access control, message packet routing, and highly configurable architecture by logical separation of roles in a network. The SIP specifications also cover a general purpose event service [Roa02]. These features make the addition of IMP services relatively simple where SIP middleware is already deployed, which makes it a very attractive IMP technology of choice for SIP adopters. Likewise the use of SIMPLE requires the presence of SIP, which is an extra overhead when deploying only an IMP solution. SIMPLE is defined as an event package for SIP events.

In the SIMPLE design [Ros04] the logical components of the IMPP reference architecture [DRS00] are specified as distinct entities. This allows for example multiple presence information producers. A SIMPLE Presence User Agent (PUA) produces information that is gathered by a Presence Agent (PA) component. The PA handles all subscriptions by Presence Clients (PCs) and sends them notifications when the presentity's state changes. Since communication is by events instead of RPC, SIMPLE facilitates terminal mobility and service reconfiguration.

SIMPLE support for mobile terminals can be criticized on some issues. First of all there is the relatively large message size. Where XMPP is able to omit session context information from single messages, SIP does not use such a transport session association, forcing redundant content in consecutive messages. SIMPLE message payload is XML, which is generally considered problematic over wireless.

The second problem is that SIP messages are typically sent using User Datagram Protocol (UDP) transport. This is a problem with many mobile terminals that are not Internet-addressable. This problem can, however, be circumvented using SIP proxies within the network infrastructure.

Secure Internet Live Conferencing (SILC)

SILC [Rii03a] is an instant messaging technology developed with security as the highest priority. In ideology it has much in common with the IRC technology but with a completely different design. Probably the motivation for the SILC design has been the various problems in IRC networks, which is apparent from the way IRC is used as a comparative reference in SILC publications. SILC relies heavily on asymmetric encryption technologies. SILC is intended to be standardized in IETF and the protocol drafts are available [Rii04b, Rii04c, Rii04a, Rii04d, Rii03c, Rii03b]. SILC has not been assigned a working group, but is being specified by the SILC Project (`http://www.silcnet.org`).

The SILC service architecture is a ring of clusters. Each server belongs to a cluster and can accept client connections. A router is a specialized server that routes messages between the servers in that cluster. Each router is connected primarily to two neighboring cluster routers and possibly secondarily to other routers in the ring. Backup routers can take over routing tasks when the primary router fails, causing only the client sessions directly attached to the failed router to be lost.

SILC connections, both client-server and server-server, are persistent and session-oriented. When connecting, the parties agree on a common session key that is used to encrypt and sign the protocol messages. By default each message is encrypted hop-by-hop. SILC also supports end-to-end encryption but setting up key management is beyond the scope of the SILC specification.

SILC developers claim mobile environment support with the compact and compressed protocol. However, the abundant use of asymmetric encryption may raise some issues. First of all is that asymmetric encryption and decryption algorithms are processor-intensive. This may no longer be an issue computing-time-wise but the CPU utilization may cause noticeable battery consumption in small devices. The impact of encryption on battery consumption might be an interesting research item.

5.2.4 Service Integrators

Some existing IMP solutions are provided by service integrators. Most notable of these are AOL, Microsoft, and Yahoo!, each with their own proprietary technologies, networks, and large user bases. What is characteristic of these kind of IMP solutions are

- close integration into other services from the same provider (e.g. email),
- ability to rapidly push innovations by the method of client updates,
- free service, either funded by advertisements or offered as an attractor, and
- rich User Interface (UI) design: the provider has in theory control over every different method of inputting and rendering information in the system[1].

Proprietary instant messengers are typically integrated to the other services offered by the service provider, at least on the level of user account management. Integration with services like video or audio conferencing has also been quite popular.

The problem with evaluating proprietary solutions is the lack of public detailed information. One approach is to deploy these solutions and then evaluate them based on hands-on experience. Consequently this review chapter focuses on open and publicly available technologies.

ICQ and AOL Instant Messenger

ICQ by Mirabilis was the first widely popular desktop presence service. The product was since acquired by America Online (AOL). Both of these provide an effective overview of the online/offline status of buddies (other states are also predefined). They also offer effective reciprocal access control and many different interaction tools (e.g. messaging, chat, file transfer). AOL is infamous for its continuing efforts to block interoperability bridging to other IMP systems.

[1]Defining a common basic state set or common rendering rules is often considered counter-productive in protocol standardization.

Microsoft Network Messenger, Windows Messenger, ThreeDegrees

Microsoft has been actively developing IMP technology in its products as well as participating in standardization work. It currently provides three different tools for what can be called presence or presence-enhanced services. Hands-on evaluation of these tools was hampered by lack of platform support. The oldest of the Microsoft tools is the Microsoft Network Messenger, largely similar to the AOL Instant Messenger. The technology in this product is proprietary and service interoperability requires a license agreement.

Windows Messenger is a more recent product, shipped as part of the Windows XP operating system. The tool apparently has a more versatile architecture and should be able to connect to different networks, using different technologies (e.g. SIP/SIMPLE). Apparently the Windows Messenger technology is being opened up in the sense that the server software is offered for service integration licensing.

The latest Microsoft product is called ThreeDegrees. ThreeDegrees integrates to the Windows XP desktop. It offers contact management and content sharing in invitation-based private persistent groups created by users. The tool has some IMP service integration with other Microsoft products and offers an interesting kind of group sharing of content like images and music. The group sharing of activity and media seem to be the key features.

5.3 Discussion

There are some open issues regarding the possibilities and different facets of presence, such as usability trade-offs, software architectural role, and online identity. Past research projects, prototypes, and user studies have provided valuable information on some of the possibilities and limitations as well as raised questions in need of answering.

5.3.1 Prototypes

Research prototypes are important in both proving new innovations and charting user reactions and behavior. The prototype listing in this section consists of only those prototypes which are seen to have raised interesting questions or provided interesting solutions.

Active Badge

One of the first notable prototypes of a presence service was the Active Badge prototype [WHFG92] in 1990. This prototype used personal badges that transmitted unique infrared signals periodically. These signals were intercepted by infrared receivers that then fed the badge spottings to the local computer network, enabling users to see where and when the badges of other people have been seen. The initial problem that this was intended to solve was that the local receptionist, when taking a call, needed to know if a requested person could be reached and where.

In [WHFG92] the authors point out issues like

- access to location data must be restricted and users should not have equal rights to all data (access control),
- the system should impose minimal overhead on the user,
- the prototyped system was vulnerable to managerial abuse, and
- a user could easily hide himself from the system at will.

At the time of the article the prototype had been in active and productive use for 16 months, and the researchers had had the opportunity to meter system usage and do user interviews, both of which indicated the usefulness and good user response of the system. The prototype managed to prove the feasibility and usefulness of a cost-effective location presence system.

NYNEX Portholes

The NYNEX portholes [LGS97] prototype was a particularly interesting attempt to provide "an overview of a community through a matrix of video images." Their prototype presented many innovations in this kind of awareness system, answering many issues about responsiveness and user privacy. The primary function of their software was to instantly determine if a given person is in his office when there is no clear line of sight. Each user terminal was equipped with a camera that took digital pictures of the person in front of it at given intervals. These pictures were then collected to a server which provided a combined view of the desks of different people.

One interesting aspect of the prototype was its unusual implementation of reciprocity: not only are permissions reciprocal but a user can get information about how often and by whom he is looked at through the portholes — "when you look you will be seen". According to their user interviews this additional reciprocity was valued, partly because the video images of self are considered more personal than a simple online/offline state. Because of this the prototype seemed to raise a lot of privacy concerns, which resulted in many different technical means of improving user control over privacy.

Hubbub

Hubbub [IWR02] is an interesting IMP system prototype for its novelties in UI functionality. Its most notable innovation is to assign a short melody for each user. When a user becomes available, this melody is played by the clients of those people who have that user in their buddy list. When a user becomes unavailable, the melody is played backwards. This way a user would learn to associate a person with a particular melody and be instantly aware of the comings and goings of that person, without any distracting visual cues popping up on screen.

The prototype also presented a couple of other interesting innovations. One was activity metering: the client making guesses on the current activity of a user on that terminal. This activity rating can then be shown on the clients of his buddies and used as a hint for routing instant messages. The content of a message was also made richer by introducing a so called "Sound Instant Message (SIM)", which is a melody of just a couple of notes denoting some pre-selected sayings like "OK", "Bye", or "Hi". In Hubbub these SIMs were used both in chat mode and as separate instant messages. In the latter case the recipient hears the sound icon of his friend and then the common SIM melody. Usability considered, it should be a clear advantage that a user does not have to separately go look who came online or who sent the message, or to be looking at a display at all to be able to receive some messages.

The research team that created Hubbub metered system usage thoroughly, using several user interviews, polls, and quite a notable analysis of the message traffic which was stored with the knowledge of the end users. Using this data the researchers made conclusions on how the system was used, typical interaction patterns, user experience, etc. Their findings suggest both work logistic and social advantages.

5.3.2 Fundamental Trade-offs

IMP technologies always introduce a tradeoff situation between privacy, awareness, and disruption [HS96]. The more detailed, and the more frequently, presence information is provided, the more effectively it can be put to use, and the less privacy the user has. Also, the more abundant the delivered presence information becomes, the more likely it is to be non-interesting and disturbing. Thus it is desirable for it to be possible to adjust these tradeoffs to suit personal preferences and different situations. The tradeoff situation can be softened by technical tools and good UI design.

If presence information is detailed and publicly available, it is very easy for people and software to make use of it. However, whatever the level of privacy required, it must be technically possible to both limit the audience of the information and the level of detail provided. To make these limitations, someone — typically the user — has to make decisions on who is given access to what information. A possibly-complex decision is then needed prior to any utilization of the information. In limited controlled communities like a workplace it might be feasible to centrally enforce the decisions both to reduce the need for decision making and policy upkeep, and to guarantee a level of awareness support within that community.

The level of disturbance an awareness system causes is a highly subjective experience. Therefore it should be possible to personally adjust the level and methods by which the awareness system intrudes to user consciousness. The level of disturbance can be affected by the UI models [IWR02] and by actively filtering out events. These of course should adapt to the current user needs: his mood, his social situation, the tasks he is occupied with. The more this adaptation can be done by reliable automation instead of direct user control is a challenge for proactive computing research. Proactivity and context awareness are also a prominent solution for making the presence data more meaningful.

5.3.3 Online Identity

The concept of online identity is not a simple one. Some systems require reliable mapping of digital identification to a real-life person, while at the other end of the spectrum the identity is only temporary, discardable, and without any reliable link to any real person. The different uses of identity could be considered to have two main attributes: the first is the reliability of the identity, the second is the lifetime of the identity. An IMP system may impose some model of identity by design, but it will be more usable if the identity model is dynamic. Furthermore, a single person has different online identities for different purposes, and likewise, several people may share the same identity. The issue of online identity is a general problem in the field of distributed applications and is yet to be resolved.

Most IMP systems require some kind of user registration. Depending on the usage conditions the user may have the choice of entering either accurate or more or less bogus information. If a user may freely give bogus information, the resulting identity can be anonymous but it is then impossible to provide a secure identity without some external authentication. On the other hand, if the system requires strong authentication for access, users may not be able to participate anonymously at all unless the system offers some methods of anonymization. In cases where the server stores buddy lists, preferences, or pending messages, it is required that a user authenticates himself to access that data: while an effective and easy solution, this excludes occasional and opportunistic use. For example, access to Microsoft Messenger service requires a Microsoft Hotmail account, while logging on to an IRC server usually requires no authentication.

As mentioned before, people have different identities for different uses and the requirements for these vary greatly. Saying “Hi, I’m Bob and I like your personality.” is of altogether different situation than “Hi, I’m Bob and would like to close my bank account.” There is a great difference in requirements depending on the situation: how reliable must the authentication be, and who should be convinced? Is reliable authentication a requirement for the development of commercial services or mission critical applications using IMP technology? Can reliable authentication co-exist with anonymous and opportunistic use?

Liberty Alliance

Liberty Alliance (LA) (`http://www.projectliberty.org/`) is a consortium of over 160 organizations that aims to define an architecture and standards for federated network identity. The design starts with the notion that online identities are facets of user identification. An identity may represent a person or a role within an organization. A single person will have multiple identities that each have their domain of significance, for example customer accounts in different vendor services. Some of these identities are casual, some anonymous, some requiring a high level of authenticity. One of the initial goals in the Liberty Alliance (LA) work is to provide a standard single-sign-on technology, enabling a user to authenticate securely to different service providers only once in a session.

The LA key objectives are listed as [CHKT03]

- Enable consumers to protect the privacy of their network identity information.

- Enable businesses to maintain and manage their customer relationships without third-party participation.

- Provide an open single-sign-on standard that includes decentralized authentication and authorization from multiple providers.

- Create a network identity infrastructure that supports all current and emerging network access devices.

Service providers maintain their own customer databases. A user can opt-in to link different, service-provider-specific, *local identities* to a *federated network identity*. The group of service providers that share linked identities and have established operational agreements that define trust relationships is known as a *circle of trust* [Lib03]. Typically a circle of trust consists of *service providers* and *identity providers* that have business relationships and with whom users can transact apparently seamlessly. Organizations offering web services to users are considered service providers. Identity providers are service providers that act as hubs of identity federation, aggregating a circle of trust.

A user may have several federated identities in different circles of trust, for example a work profile for an enterprise circle of trust and a home profile for some consumer circle of trust. Authenticating to a circle of trust typically enables automatic authentication to all service providers in that circle. This authentication happens on demand. The LA specification allows service providers to have different requirements for authentication reliability and credentials required: when using a stricter service the user may be asked for stronger proof of identity.

By default service providers have a very limited amount of information about the other accounts of a user: typically only the identity of the user in the other service in an identity federation. An identity provider can also create an anonymous federation where the service provider doesn't even see the actual identifier of the user in the identity provider's domain. An identity provider can disclose properties of a user identity, such as address, payment abilities, or full name, to other providers but must adhere to the user privacy policies.

The identity providers — the hubs in the circles of trust — facilitate the scalability of the LA architecture since authentication can be delegated from one identity provider to another (if the two have a mutual trusted business relationship). This way a circle of trust is not a rigid pre-determined collection but an emergent network of service providers. This is both an administration advantage and a security risk [PW03].

The LA specification[1] defines different protocols. The basic client-provider protocol uses the functionality of a typical web browser, taking advantage of HTTP redirection and possibly SSL and ECMAScript capabilities. This has the advantage of not requiring any client side upgrades to take advantage of the LA identity federation. There is also a separate client-provider protocol that provides LA-aware applications a richer set of operations and account management. The communication between service providers uses a separate SOAP-based protocol.

5.3.4 Embedded Presence

Many existing presence applications are instant messaging systems, dedicated to the explicit listing of associates's states and sending them messages. We are now beginning to see embedded presence applications: applications that enhance their functionality by making use of available presence information. A prime example of this is a telephone contact list that is able to show the states of the contacts if there is presence information available on them. In a similar way presence information cues can be useful in other collaboration or communication tools: an email client showing the state of the sender and the recipient, distributed edit and sketching tools showing the activity of other participants [HD02], a work task management interface showing the readiness and location of available personnel, etc. In essence, the presence information is offered embedded within the application in the context of the task at hand.

With the advent of IMP standards, the feasibility of these kind of implementations is becoming better. There may arise a need for tools on popular platforms for leveraging these standards through a uniform and specialized API, making it easier to embed presence information to various pieces of software.

5.3.5 Related Research

The current main active focuses in relation to presence are on context awareness, usability, social issues, and different collaboration tools. Since finding ongoing research projects through publications is not easy, the projects mentioned here do not present any conscious selective set.

The Context project (`http://www.hiit.fi/fuego/context/`) at the Helsinki Institute for Information Technology uses presence information to

[1]`http://www.projectliberty.org/specs/`

study models of automatic user context classification and proactive recognition. The methodology used includes “qualitative user studies, data analysis algorithm development, and empirical testing in a prototype environment”. The project was launched in November 2002 and is scheduled to last until the end of the year 2005.

Adaptive and Context-Aware Services (ACAS) (http://psi.verkstad.net/ACAS/) at the Wireless Center of the Royal Institute of Technology in Stockholm studies the areas of “Ad hoc service environments”, “Seamless adaptive personal services”, and “Smart adaptive media”. These cover such interesting topics as user context management, modeling and ontologies, proactivity, access management, mobility, and software component architecture.

5.4 Conclusions

A mobile presence service intersects with many interesting and important research areas. Integration of presence services with applications requires solutions for supporting a network where interconnected heterogeneous terminals publish presence information for each other. A presence service is a promising enabler for proactive and context-aware tools and collaboration applications, but several issues remain to be addressed. Globality calls for interoperable standards, ubiquity calls for reliable privacy, mobility calls for attention on heterogeneous environments and adaption to computing context. The work regarding presence in the Fuego Core project has created a very basic distributed presence service architecture and software component interface. This prototype will be extended and elaborated to a presence and context framework for possible application development in later projects.

Chapter 6

Host Identity Protocol

6.1 Introduction

The Internet protocol suite was originally designed with the assumptions that hosts are static and trusted. These assumptions are not valid anymore. Mobility has become increasingly popular. The number of users on the Internet has grown exponentially. Everyone does not know everyone anymore and thereby cannot trust everyone. Some extensions to the existing protocol suite are needed to keep up with the changes in the nature of the Internet.

There have been many engineering efforts to redesign the Internet protocol suite to meet the new requirements. SCTP [KMP+00] and Mobile IP [Per96] are a few examples of these efforts. Host Identity Protocol (HIP) [MNJH04a] is a protocol among many others to extend the current protocol suite.

This chapter introduces the background motivation for HIP and presents a short overview of the HIP architecture. Related protocols, such as Mobile IP, will be shortly presented. The current status of HIP will also be reviewed.

6.2 Background

The concept of an endpoint identifier has been long neglected [Chi99] by the Internet community in network and transport level protocols. Endpoint identifier means the ultimate name of the end of a transport level connection, and it is decoupled from the location of the endpoint. Neglect of endpoint identifiers involves everyone using the Internet.

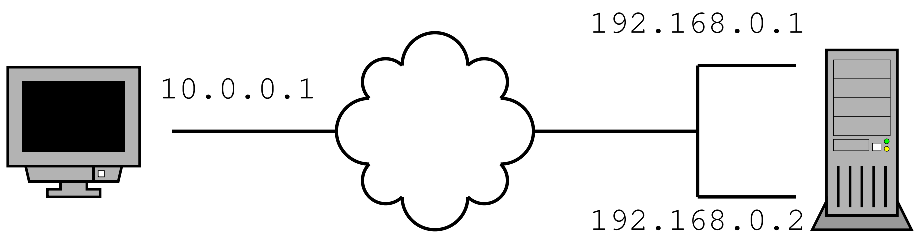

Figure 6.1: A simple end-host multihoming example

To illustrate the problem with endpoint identifiers, let us consider two different scenarios: multihoming and mobility. The end host acts as a traditional "client" using a light-weight terminal or a personal computer. The end host accesses the Internet through a home network, which is some kind of IP-based residential or wireless network.

In general, multihoming means that there are two or more different routes to a destination host. The redundant paths can be used in protecting against network failures, enabling load sharing, and improving performance [ABG02].

Multihoming can further be divided into two different types: host multihoming and site multihoming. Host multihoming means that the host has two or more interfaces to the network, as in Figure 6.1. Site multihoming means that the site has multiple connections to other networks, and hosts in the network have only one connection to the network. An interested reader should see [ABG02] for more information about site multihoming.

Mobility means that the client changes its topological location in the network (Figure 6.2). This usually means that the client has to change its IP address. The IP address can change for various reasons, e.g. if the host roams to another network or its Dynamic Host Configuration Protocol (DHCP) lease expires. Mobility also means that existing connections should not be torn down, although a small delay is usually involved in changing the IP address in a real-world situation.

Transport level connections in the Internet are formed commonly with Transport Control Protocol (TCP) [Pos81] or UDP [Pos80]. Both TCP and UDP use IP addresses as a part of their endpoint identifiers, that is, they share the same address space. The reason for sharing the same address space was a design decision based on the requirements of networks when the Internet was still highly under development: the hosts were quite static and there was no need to add a new address space for transport level connections. The new address space would have been redundant, because the hosts were static, and it would have involved an additional burden for routing. Routing would have been more complex because a mapping of

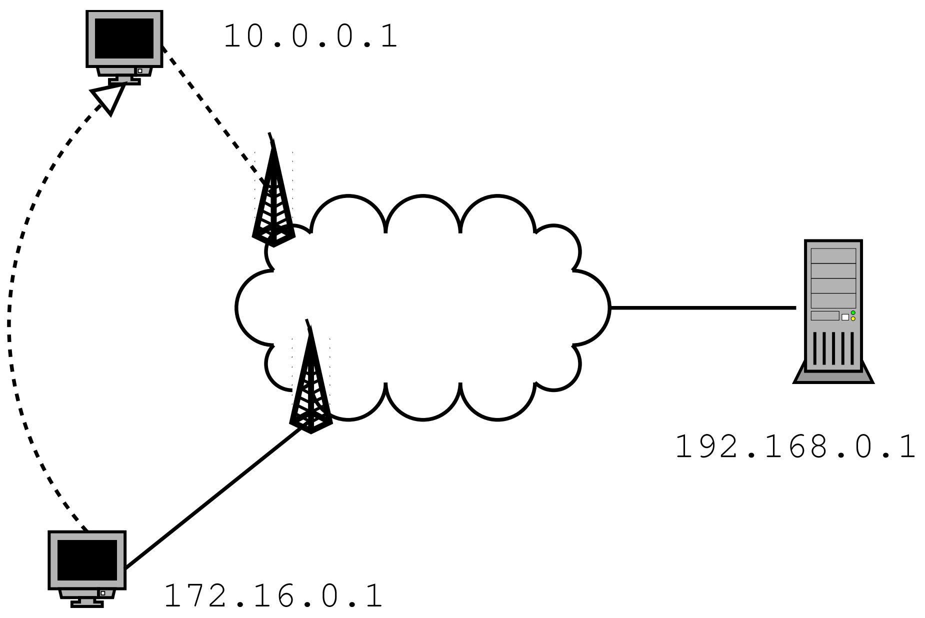

Figure 6.2: A simple mobility example

identifiers from the transport level to the network level (and vice versa) would have needed to be introduced.

Using the same address space at both the network level and the transport level has introduced problems in multihoming and mobility [Chi99]. Here are two examples from the IPv4 world to further illustrate the problems:

End-host multihoming There is a client (marked with a monitor) with one network interface (10.0.0.1) and a server (marked with a PC tower) with two network interfaces (192.168.0.1 and 192.168.0.2) in Figure 6.1. Consider a situation where the client has established a transport level connection with the server and the connection is routed from 10.0.0.1 to 192.168.0.1. Now, the interface on 192.168.0.1 breaks down for some reason. The obvious solution would be to reroute the connection via 192.168.0.2, but this would break the connection in IPv4 or IPv6 without some kind of mobile IP support.

Mobility If a host moves to another network (Figure 6.2), it has to change its IP address. Again, all transport level connections will be torn down when the IP address changes, because the connection is bound to a static IP address.

Both of the previous examples have a common problem: changes in network level routing also affect end-to-end transport-level connections.

The reason for this is that the transport level uses the same identifiers (IP addresses) for hosts as the network level. This causes a problematic dependency for the transport level on the network level and is one of the major reasons for multihoming and mobility problems.

There are many solutions and workarounds for multihoming and mobility problems. Mobile IP [Per96] and SCTP [KMP+00] have been engineered to tackle the problems, just to mention a few alternatives. HIP is one of many alternatives. It offers a new namespace for identifying hosts independently of network-level routing, thus solving many of the mobility and multihoming problems. HIP also uses public-key cryptography to reduce various network attacks and to make internetworking safer.

6.3 Architecture Overview

6.3.1 Host Identity

Endpoint identifier issues were shortly introduced in section 6.2. The endpoint identifier in HIP is called a Host Identifier (HI). The networking stack of the host has one or more HIs that are used for identifying transport-layer endpoints in a location-independent manner. Two types of HIs exist: "well-known" public HIs (can be published in Domain Name Service (DNS) or in some other known place) and anonymous.

HIs are permanently integrated with security, because they are cryptographic in nature. The best HI is the public key part of an asymmetric key pair. HIs based on a public key can be used in HIP packet authentication and for protection from man-in-the-middle attacks. A HIP implementation must support at least Digital Signature Algorithm (DSA) [NIS94] as a cryptographic algorithm for HIs

Every host typically has at least one HI. It is, however, recommended that each host has also an anonymous HI that it can use if needed. HI public keys could be stored using Light Weight Directory Access Protocol (LDAP) [HM02] or the DNS [Moc87]. Anonymous HIs should not be stored in the DNS (otherwise they would not be anonymous). HIs are potentially long, so they are not used directly in Internet protocols.

HIs are long because asymmetric cryptographic algorithms tend to use long key material. Therefore HIs are not suitable for socket API calls or packet source and destination addresses. A more compact representation for HIs in HIP is described in subsection 6.3.2.

6.3.2 Other Representations of Host Identity in HIP

A HIP protocol implementation using only HIs would be inefficient because public keys tend to be long and inserting a long public key into a packet causes too much overhead. A public key could also be of variable length and support for variable-length public-key identifiers would be harder to implement. Instead, a hash over the public key is used, producing a 128-bit field called a Host Identity Tag (HIT).

A HIT identifies a HI in an efficient way and can be used for further negotiation between the end hosts. A HIT should be statistically unique but collisions are still possible. A HIT should be interpreted as a hint of the correct public key in a collision situation.

A Local Scope Identifier (LSI) is even shorter than a HIT. A LSI is a 32-bit localized representation of a HI. LSIs exist mainly to support backwards compatibility with the IPv4 API.

The fixed lengths of LSIs and HITs correspond exactly to IP address lengths in IPv4 (32 bits) and IPv6 (128 bits). This is not at all a coincidence but a careful design choice to support existing TCP/IP architectures. Fixed length eases protocol coding and control of packet sizes, and the format is suitable for underlying protocols despite the identity technology used.

For example, to make a TCP connection to a host using IPv6, one must create a socket and connect the socket to the peer host's IP address and port. The connection mechanism and API calls could be transparently the same in HIP-aware end hosts, but the IP address would be interpreted as a HIT. This means that the semantic meaning of the API call is changed in HIP.

6.3.3 Base Exchange

Before HIP connections can be established between hosts, they need to be authenticated. This is the main purpose of the base exchange. During the connection establishment procedure information required for creating IPsec Security Associations [KA98a] is also shared.

After the base exchange is finished successfully, both hosts have authenticated themselves to each other and set up their security associations. After this, the connection continues using the newly created IPsec Security Associations.

The base exchange is similar to the TCP connection establishment procedure [Pos81]. TCP has a three-way handshake to establish state between two hosts. The HIP protocol needs four packets to establish state between two hosts. The base exchange has been carefully designed to avoid possible denial-of-service and man-in-the-middle attacks. TCP and UDP [Pos80] also have protection when HIP enables the use of Encapsulating Security Payload (ESP) [KA98b] via Security Associations.

6.3.4 Security and Privacy

The HIP protocol brings security into the existing TCP/IP architecture. First, the end hosts are authenticated using the base exchange. The public keys of the hosts can be verified from the DNS. The connection is encrypted with ESP after the authentication is done. Privacy issues are handled in the protocol as well. Anonymous HIs [MNJH04a] are specified to provide almost anonymous access to the network.

6.3.5 Multihoming

A multihomed host has more than one network interface, each of which is connected to a possibly-different network. These interfaces therefore have different addresses. Due to this difference in addresses, a multihomed host seems to be located in different locations within the network. Its operating system keeps track of the mapping between an address and its corresponding interface.

End-host multihoming can be attained quite simply using HIP. In the HIP namespace, mappings are created between IP addresses of all interfaces and a single HIs. Every IP address is mapped to the same HI. IP addresses present a topological location within the network, and the HI is used as the endpoint. Now the multihomed host looks like it is located in one place when communicating using HIP.

6.3.6 Mobility

There are different types of mobility, such as network and end-host mobility. This section concentrates only on end-host mobility issues with HIP.

The purpose of mobility becomes clear when an end host changes its IP address after it has roamed to another network, for example. Before the roaming, all transport layer connections between the end host and its peers are bound to IP addresses in the previous network. Effectively this means that the live connections are torn down because the IP packets associated with the connections are routed to the wrong network and IP address. End-host mobility support can prevent this kind of situation and keep the connections alive.

Mobility in HIP is quite straightforward if Dynamic Updates in the Domain Name System (DNS UPDATE)[Wel00] is available [Mos01]. The mapping of HIs and IP addresses is updated rapidly in the DNS and all packets will be routed to proper IP addresses. Difficulties arise when DNS UPDATE is not used and mobility management has to be handled differently.

Mobility can be divided into three different categories from the HIP point of view depending on the party who changes its IP address. Initiator and responder mobility will be discussed in this section. The term “initiator” refers here to the host initiating a connection to a peer host. The peer host that will receive the connection attempt will be referred to as the “responder”. The third kind of mobility is about the double jump problem and it means that both end hosts change their IP addresses at the same time. The double jump problem is not discussed here and a reader interested in the topic should refer to [Nik02].

Initiator mobility is simple. The responder can accept a HIP or an ESP packet, whose Security Parameter Index, SPI [KA98a] is known to HIP, from anywhere. The IP address is then used only for routing the packets back. The initiator may also send HIP UPDATE packets containing its new addresses in a REA parameter to the responder to tell the initiator’s new location.

In responder mobility, the initiator needs to know the location of the responder. This method uses a packet forwarding agent which is also referred to as a rendezvous server in the literature. The packet forwarding agent acts as a virtual interface of the responder [Nik02], representing the IP address of the responder. Initially, the responder sets up a HIP-based Security Association with the packet-forwarding agent. The responder updates its IP address continuously to the packet-forwarding agent using HIP packets containing readdressing (REA) parameters.

The initiator sends its initial HIP packet destined to the responder’s HIT. The IP address in the packet is not the actual IP address of the initiator but the IP address of the forwarding agent. The forwarding agent forwards the packet to the responder’s current location. After this packets are exchanged directly between the initiator and the responder, and responder mobility is handled like initiator mobility. The packet-forwarding agent is discussed in more detail in [Mos01].

6.3.7 Rendezvous Server

In order to start the HIP exchange, the initiator node has to know how to reach a mobile node. Although Dynamic DNS could be used for this function for infrequently moving nodes, an alternative to using DNS in this fashion is to use a piece of static infrastructure called a HIP rendezvous server. Instead of registering its current dynamic address to the DNS server, the mobile node registers the address(es) of its rendezvous server(s). The mobile node keeps the rendezvous server(s) continuously updated with its current IP address(es). A rendezvous server simply forwards the initial HIP packet from an initiator to the mobile node at its current location. All further packets flow between the initiator and the mobile node. [MN03]

The current situation is that there are no publicly known implementations of the rendezvous server. In practice, this means that its specification might change a bit, and thus it remains an open issue for the time being. L. Eggert presented in [Egg04] different scenarios and variations of the rendezvous server. The primary point of concern in his Internet draft is, how do non-HIP-enabled and HIP-enabled hosts interact without prior knowledge of each other's capabilities?

It is expected that this part of the HIP infrastructure will change or will be clarified during this year. As the base HIP draft stabilizes, work will concentrate more on the infrastructural elements of the HIP architecture.

6.3.8 Native Application Programming Interface

Nobody has shown a complete specification for an application programming interface for HIP. Partial work has already been done [Mos01] in the form of a legacy API. The legacy API requires only minor changes in applications and therefore it cannot utilize all the properties of a HIP-enabled networking stack. Work on the "native HIP API" [Kom04] has been finished at the Helsinki Institute for Information Technology, and the results of the experimentation will be used to form a draft for the IETF.

The most significant change of the native API is the fact that it uses public-key identities in the user-space socket API. A direct benefit of this is that the users can provide their own public key identifiers to the networking stack. This means that the identities are no more bound to just hosts; they can be bound to users, processes, or groups. There are also plans behind the scenes to state in the next version of the HIP drafts that DNS should really store public keys instead of HITs, so the explicit public key handling in the native HIP API may come in handy.

The native API also utilizes the HIP layer better than the legacy API. Anonymous identities, opportunistic HIP, and certificate handling can be supported explicitly from the native API. Quality of Service can also be controlled with unified policy management interfaces. Some degree of multi-protocol compatibility has been planned for the native API to support other protocols that make the separation between endpoint identifiers and locators like HIP does.

6.3.9 Advantages and Disadvantages

In this section, we list both the advantages and disadvantages of HIP shortly. Let us begin with the advantages:

- HIs provide consistent names for hosts regardless of how they connect to the Internet.
- Cryptography-based public-key-like names are difficult to steal.
- HIP separates the routing and host namespaces with no effect on routability. The role of the IP address changes to simply a packet-forwarding namespace.
- The protocol is very simple and bandwidth-conservative; there is no per-packet overhead beyond that of IPSec after the Security Associations (SAs) have been established.
- The base exchange provides a secure authentication for the hosts.
- Both the key exchange and the initialization of IPsec security associations for use with HIP are relatively lightweight. This may reduce the need for complex external infrastructures such as Internet Key Exchange (IKE) [HC98] even though HIP does not provide all the same functionality as IKE.
- Forwarding at rendezvous server does not need a lot of processing power (only a couple of packets per request). One server can easily handle a large number of hosts.
- HIP implements a “stateless” connection handshake to provide denial-of-service resilience [Hen03] especially against malicious initiators.
- HIP also provides privacy support [Hen03] in the form of anonymous identities.

- HIP is tightly integrated with IP security protocols [Hen03].
- The HIP solution for multihoming is more natural than many other existing ones [Hen03].

Potential disadvantages of HIP are

- Decoupling of internetworking and transport layers and adding a new host layer might require changes to the existing APIs and applications.
- Locally created anonymous HIs make resolvability very difficult for other hosts.
- Asymmetric cryptography consumes the computing capacities of end-hosts. This presents a problem for low-performance devices such as handhelds and mobile devices [Kom02].
- Changing HIP to support multicast may require a substantial amount of work.
- Dynamic binding of HITs to IPv6 addresses may have some security vulnerabilities that haven't been found yet [Nik02]. The forwarding agent could be another source of potential vulnerabilities.
- Mobile IP and other related protocols have been around for a while and they have the advantage over time.
- HIP requires either dynamic DNS updates or additional network infrastructure to allow hosts to find mobile servers [Hen03].
- For practical use, HIP requires Public Key Infrastructure (PKI) or extensions to DNS, unless operated in anonymous mode [Hen03].
- HIP requires changes to networking stacks and APIs at both ends of the connection [Hen03].
- Unless applications are HIP-aware, HIP has problems if IP addresses are used in the application data stream [Hen03].
- DNS-centric approaches to end-host mobility require a reduction in the TTL for records for mobile nodes, and hence a greater load on root nameservers [Hen03].

- If HIs are used to identify hosts, the loss of host aggregation based on network prefix could cause scaling problems for access control lists containing a network address and mask [Hen03].
- There is little implementation and operational experience [Hen03].
- The overhead caused by the handshake is high for short transactions [Hen03].

6.4 Related Work

Some protocols have some of the same kind of functionality as HIP has. Examples of these protocols are Stream Control Transmission Protocol (SCTP) and Mobile IPv6.

6.4.1 Mobile IPv6

Mobile IPv6 tries to make mobility transparent to applications using higher-level protocols. Mobile IPv6 also addresses security issues [Per96].

Mobile IPv6 can be seen as an alternative to HIP and vice versa although both Mobile IPv6 and HIP tackle somewhat different problems. Mobile IPv6 does not currently address end-host multihoming, and it is still trying to provide a scalable security solution [NYJW04, YJWN02].

6.4.2 MobIKE

Mobile IKE (MobIKE) protocol [Kiv04] tries to enchance IKE version 2 with mobility support. The main scenario concerns Virtual Private Network (VPN) users moving from address to another without re-establishing SAs as stated in the IETF working group description. Some co-operation with the SCTP is also planned.

6.4.3 SCTP

SCTP provides a means for each SCTP endpoint to provide to the other endpoint (during association startup) a list of transport addresses (i.e. multiple IP addresses in combination with an SCTP port) through which that endpoint can be reached and from which it will originate SCTP packets. The association spans transfers over all the possible source/destination combinations that may be generated from each endpoint's lists [KMP+00]. Connection setup with SCTP is similar to HIP: a cookie exchange mechanism is also used during the connection setup procedure. SCTP uses cryptographic hash functions for data integrity checks.

6.5 Current Status

6.5.1 Standardization Status

Robert Moskowitz, the original author of HIP, has released three Internet Drafts on HIP: [MNJH04a], [MN03], and [Mos01]. They describe the HIP architecture, namespace issues, and implementation aspects of HIP. Later, Pekka Nikander has updated the draft [MNJH04a] to newer versions. As of this writing, the latest draft [MNJH04a] is managed by a new editor, Petri Jokela. The latest draft also has a new author, Thomas Henderson. The mobility and multihoming draft [NA04a] was updated.

Two HIP-related working groups were formed in the IETF in 2004. The HIP WG concentrates mainly on the development of the current drafts, emphasis being on the base specification which is one of the working group items. The other working group is HIPRR, the HIP Related Research working group. HIPRR deals with ideas developed further from the base specification, such as using Distributed Hash Tables within HIP, for example. HIP does not have any official RFCs yet.

In July 2004 L. Eggert released an updated draft of *Design Aspects of Host Identity Protocol (HIP) Rendezvous Mechanisms* [EL04b]. Quoting the draft "it discusses design aspects of rendezvous mechanisms for the Host Identity Protocol (HIP). Rendezvous mechanisms, such as HIP rendezvous servers, improve operation when HIP nodes are multi-homed or mobile. They can also facilitate communication between HIP and non-HIP nodes. Possible rendezvous mechanisms differ in performance, compatibility, and impact on the HIP and Internet architectures."

A draft by L. Eggert on rendezvous extensions was also released, *Host Identity Protocol (HIP) Rendezvous Extensions*[EL04a] in July 2004. The draft discusses rendezvous extensions for the Host Identity Protocol (HIP). Rendezvous mechanisms extend HIP for communication with HIP Rendezvous Servers. The draft motivates the need for rendezvous mechanisms, and it describes the protocol extensions in detail.

An initial draft on HIP NAT issues was published in February 2004. The draft, *Problem Statement: HIP operation over Network Address Translators*[SQ04] tries to covers issues when HIP is used between different address realms having Network Address Translators (NATs) between them. Two main areas in the draft are HIP communication across NATs and HIP-initiated IPsec-based data transmission across NATs.

6.5.2 HIP Projects

HIP is currently being developed in five independent projects. The Host Identity Protocol in Linux (HIPL) (`http://www.gaijin.iki/hipl/`) project currently has two members at the Helsinki Institute for Information Technology. Andrew McGregor is developing a cross-platform version (`http://www.sharemation.com/adm01bass/pyhip/pyhip-2003-11-15.tar.bz2`) in the Python programming language, but due to his time constraints the development has been slow and the implementation is not up to date with the latest specifications. Ericsson Research is working on an implementation for NetBSD (`http://hip4inter.net/`), Boeing is developing a Linux implementation, and Julien Laganier at Sun Microsystems in currently working on a BSD implementation.

6.5.3 Meetings

Interoperability Tests prior to IETF

HIPL has been in co-operation with Ericsson's HIP for NetBSD Project team from the beginning of the implementation work. Several interoperability tests have been performed to test each project's own implementation against the other implementation in order to prove the implementations correct and gain more experience on the subject. During the interoperability tests several ideas were raised on the problematic areas, and some good ideas were even incorporated into future drafts. Interoperability tests have been done in approximately every 3–5 month interval.

HIPL has also done some short interoperability tests with Julien Laganier from Sun Microsystems in late 2003. Sun's implementation was still in very early development at that time, so the tests were not fully successful, but they were still promising.

55th IETF

The 55th IETF meeting in Atlanta in November 2002 proved that the HIP specifications are feasible to implement. Many different HIP implementations were able to establish the base exchange almost completely, even though some workarounds had to be made because some severe bugs were found during the testing. Projects and persons involved in the testing were HIPL, Ericsson, Boeing, Andrew McGregor (Indranet), and Tim Shepard. HIPL had only an IPv6-based implementation, Boeing had only an IPv4-based implementation, and Ericsson and Andrew McGregor had support for both the IPv4 and the IPv6 protocols.

56th IETF

Between the 55th IETF meeting and the 56th in March 2003 the main purpose of the HIPL project was to fix the bugs found in the 55th meeting. Most of the bugs were fixed, so testing during the 56th meeting was more straightforward than in the 55th. Only some small details were left to be tested when the 56th meeting was over. HIPL could test ESP connections only with NULL crypto enabled, Ericsson succeeded with some other algorithms than NULL crypto. It was also noticed that the interest towards HIP is growing.

During the 56th IETF meeting, the HIP-related mailing list[1] has received more than two or three times the amount of messages compared to a year ago. The HIPL group also discussed an API proposal with the Boeing group but no consensus was established because of contradicting views. Discussion about the API will continue and merging the best of both views seems to be a reasonable solution.

57th IETF

In addition to HIPL only one of the HIP implementors, Andrew McGregor, was present at the meeting. The interoperability tests concentrated on the HIP base exchange and were successful. Some issues on the HIP native API were also discussed with Andrew McGregor and Tim Shepard from the Massachusetts Institute of Technology.

58th IETF

The first HIP Birds of a Feather (BoF) was held in the 58th IETF. The goals were to introduce the current status of HIP and discuss forming a working group. The HIP base protocol was considered more or less ready, with more work needed on infrastructure issues.

[1] http://lists.freeswan.org/pipermail/hipsec/

The BoF included a demonstration of the Ericsson and HIPL implementations. The main purpose of the demonstration was to show that the base exchange works between these two different implementations. This setup used IPv6. The demo was successful. Ericsson also had a mobility and application interoperability demo using their own implementation.

Boeing gave a presentation “Potential Applications of HIP at Boeing” which briefly described their plans of possibly using HIP all over their intranet. Their goal was to test HIP deployment on a larger scale than it has been tested so far.

Forming of the HIP WG was confirmed. The scope of the proposed charter was perceived as too ambitious for a working group. Several people voiced the opinion that the working group should focus on making a minimal usable solution, or a core solution, and the rest of the work should probably be moved to an IRTF research group.

59th IETF

Interoperability tests with Sun and Boeing implementations were successful. Andrew McGregor from IndraNet did not participate.

Miika Komu had unofficial discussions on his HIP native API topic. He got feedback on the current early API implementation experiences.

Two HIP-related events were scheduled in the 59th IETF: Host Identity Protocol BoF (HIPBOF) and HIP Related Research Group (HIPRR).

The planned HIPBOF agenda had the following items:

- draft and issue status (adopt only the base specification as a WG draft at this point of time)
- WG quality control plan
- Open issues in the base specification
- Resynchronization
- Reject mechanism
- Open issues in the mobility and multihoming specification
- Return routability test

The HIPRR agenda had the following items:

- Current RG status

- HIP and related research elsewhere
- Potential research areas
- Coverage of specific topics

60th IETF

The 60th IETF was held in San Diego. The agenda for the IETF consisted of interoperability testings, and the working group and research group meetings.

Interoperability Tests Interoperability tests of the HIIT HIP implementation against Ericsson's and Boeing's implementations were successful. Andrew McGregor's Python-based implementation is badly outdated and there is no guarantee that his implementation will be updated in the future. Julien Laganier (Sun Microsystems/INRIA) did not have the time to update his implementation for the meeting.

It is possible that HIPL may have interoperability tests against a Windows implementation some time in the future if one will be available. Boeing has initially planned to start implementing HIP for Windows.

HIP Working Group Meeting Petri Jokela, the editor of the base draft, began with a discussion of the base specification draft [MNJH04b]. The specification has matured because only a few minor technical changes were introduced in the latest version. Erik Nordmark proposed a solution to the problem in which a host loses its state when it boots. This proposal will be added to the next version of the draft.

Another change was also suggested to the base draft. Albeit this occurred after the meeting, it is worth mentioning here. Pekka Nikander also suggested another change to the base draft in the mailing list. He suggested making HIP SIGMA [Kra03] compliant to ensure the security properties of the HIP base exchange.

Thomas Henderson, a chair of the HIP research group, continued with the topic of "making HIP multi6 friendly". The IETF Multi6 working group is designing a site multihoming solution based on IPv6. One of the goals of the multi6 working group is that the solution should not have weaker security properties than the current Internet design has. As work for the solution is at its initial stages, the working group has also considered HIP as a basis for their solution because of the similarities in the goals of the working groups. The security properties of HIP are relatively strong, and perhaps even too strong from the viewpoint of multi6, so a light-weight-HIP proposal [NH04] has been written. However, multi6 compatibility will not be considered in the base specification for the time being.

Miika Komu from HIIT gave a presentation on the legacy HIP socket API. The nature of the presentation was informal. Its purpose was to prepare IETF participants for the following HIP research group session, which included a presentation of an extended HIP socket API. In short, the legacy HIP socket API is a backwards compatible socket API for legacy applications. Legacy applications, in turn, are those applications that are difficult or even impossible for their developer or maintainer to modify in order to make them use HIP.

The legacy API was divided into three categories: referral, transparent, and native API. The referral-based API is the most backwards compatible API, as the userspace Application Identifier (AID) is an Internet Protocol (IP) address. This way, applications can safely inform each other about their IP addresses. FTP-based applications are a typical example of this category. The transparent API is somewhat similar to the referral API, as the maintainer of the application does not have to modify the application code at all. However, the AID is a HIT instead of an IP address in the transparent API. In a way, the transparent API is an intermediate form of the referral and native APIs. The application developer must update the source code in order to enable HIP in the application. After the update, the application is HIP-aware, and thus can control the HIP layer better.

Thomas Henderson continued with a discussion on the HIP mobility and multihoming draft [NA04a]. First the design goals of the updated draft were presented: session persistence across multiple locators, address verification, possibility of locator change with or without rekeying, concept of a preferred address, announcing additional locators in the base exchange, and middle box issues. The draft does not cover rendezvous servers or transport layer issues. The most notable change in the updated mobility draft is the use of the UPDATE mechanism instead of the old style REA packets. Also the concept of address group was removed. Implementation experience would help to solve some open issues left in the draft, and indicate if the proposed design is feasible to implement. HIP for Linux (HIPL) is planning to implement some level of support for this new mobility draft. It is hoped that this will help to understand possible design flaws and propose better ways to improve the specification.

The effort for specifying rendezvous mechanism was split into two. Legacy rendezvous mechanisms [EL04b], i.e. from a HIP-enabled node to a non-HIP-enabled node is a work in progress in the research group.

In the working group session, Julien Laganier presented the HIP-to-HIP rendezvous mechanism draft [EL04a] for the first time.

The main purpose of the HIP-to-HIP rendezvous server is to forward initial I1 packets originating from a correspondent node to the mobile node. As such, it provides a faster alternative to dynamic DNS updates because the only IP address needed to configure to the DNS is the address of the rendezvous server, which is typically long-lived. The communication between the mobile node and the rendezvous server can be protected by using HIP itself. Later, the full HIP SA can be transformed into a "soft association" that does not consume as much resources as a full HIP SA.

The rendezvous extensions needs more implementation experience. The HIPL implementation at HIIT already has an experimental implementation of a HIP-to-HIP rendezvous server.

Julien Laganier continued with the HIP DNS extensions draft [NL04]. He proposed some technical enhancements on the objects that can be stored in the DNS. For example, full HIs can be stored in the DNS instead of just HITs. Also, distinguishing rendezvous server locators from "normal" host locators is possible because the rendezvous server locators are stored in their own dedicated Resource Records (RRs).

In the HIP DNS extensions presentation, Julien Laganier proposed some technical questions to the HIP community. The exact format of the identifiers and locators is still an open issue. Especially the relationship between a HI and a locator is many-to-many in the current scheme provided by the draft, but it would be possible to constrain this relationship with different kind of RRs. For example, the locators can be indexed with HITs or sequence numbers to fix a HI to a certain locator. In effect, this would change the relationship between a HI and locator to one-to-many.

HIP Research Group Meeting The HIP research group session began with a presentation by Miika Komu from Helsinki Institute for Information Technology (HIIT) and Julien Laganier from Sun. The topic was native APIs for HIP [Kom04]. To use the APIs, developers have to modify the applications. As a consequence, the applications become HIP-aware, and can control the HIP layer better. For example, it is possible for the applications to provide their own identities to the host.

Martin Stiemerling from NEC Europe Ltd. continued with a discussion of the HIP relationship with NATs. The use of NATs was not promoted because HIP does not work well with NATs; either one or the other needs to be modified. HIP may have to be run on top of UDP in the future when IPv4 is being used.

Lars Eggert from NEC gave a presentation of the HIP rendezvous draft [EL04b]. The draft focuses on the possibilities of rendezvous servers, i.e. it is not on a technical level yet. Marco Liebsch from NEC has written some sections to the draft related to location privacy. The draft still needs more feedback from HIP community.

Michael Walfish from MIT presented some ideas of a layered naming architecture [BLR+04]. In the proposed architecture, the identifier namespaces are flat, and there are more identifier layers than in the current Internet architecture. For example, there are user-level descriptors (e.g. email addresses, search strings), service identifiers (e.g. HTTP) and endpoint identifiers (e.g. HI). The new layering structure would benefit from HIP mobility and multihoming. Also, it would make the co-existence with middle boxes easier. For example, demultiplexing with NATs would be based on endpoint identifiers rather than overloaded port numbers.

Karthik Lakshminarayanan from University of California, Berkeley presented an overview of Internet Indirection Infrastructure (i3) [SAZ+02]. It is a forwarding infrastructure that allows users to control routing and naming. i3 also provides a lookup service based on DHTs. It provides support for mobility, multicast, and anycast. NAT traversal works transparently with i3 as does secure VPN access. Security is not forgotten either: packets can be routed through an Intrusion Detection System (IDS) node, and protection against certain types of Denial of Service (DoS) attacks is provided too.

Jari Arkko from Ericsson gave a presentation on Host Identity Indirection Infrastructure (Hi3) [NA04b]. The work is motivated by the identifier and locator split in HIP: how can a host use only a HIT to initiate end-to-end network connections without the IP address?

HITs are not resolvable from the DNS, but DHTs can support resolving of HITs to IP addresses. i3 is an overlay based on DHTs. HIP can utilize i3 as a lookup mechanism, which also solves the referral problems. HIP can also be enhanced with protection against some DoS attacks, like I1 storms, due to the distributed nature of i3.

In the proposal, i3 is used for transporting the HIP control messages. Regular IPsec is used for carrying the data traffic without i3. The approach still works with middle boxes as they can learn about the traffic from the Security Parameter Index (SPI) numbers in the ESP headers.

6.6 Conclusions

Mobility, multihoming, and security have been an active research area in recent years because the current Internet architecture has not been originally designed with mobility and security in mind. Many alternative solutions have been engineered to address the problems with the current design, and one of them is HIP.

HIP would be an ideal alternative solution from the point of view of consumers, because it is almost a complete “all-in-one” system: security, mobility, and multihoming are included. HIP still needs some auxiliary systems, like forwarding agents, for full mobility. HIP also needs changes in networking stacks of existing end hosts, and may also require changes in the middle boxes.

Performance measurements and interoperability testing have shown that HIP can really be made to work in practice. Several HIP implementation projects have been developing their independent prototypes. There has been growing interest in the HIPL implementation, especially from the beginning of the year 2004. Several people have been in contact and interested in the HIP implementation at HIIT.

Work is in progress to overcome the problems with NATs. A very important research topic is to evaluate the effects of HIP deployment on a large scale, and to see how the identity-locator split changes the nature of the Internet.

In the future, the security and identifier namespace properties of HIP can even be enchanced by combining it with i3. Support for HIP multicast may also be easier to design with i3.

Chapter 7

SIP and Events

7.1 Introduction

Session Initiation Protocol (SIP) [RSC$^+$02] is a standard text-based application-layer signaling protocol. The SIP protocol aims to meet the requirements of the telecommunications industry by converging with software technologies. The applications of SIP vary widely from Internet telephony to control applications, including e-commerce, multimedia conferences, and instant messaging services. Also, the SIP protocol is considered to be a valid option for home networking. Intensive research is being conducted by the Internet Engineering Task Force (IETF), concentrating on various SIP related topics: SIP [RSC$^+$02] itself, SIMPLE (`http://www.ietf.org/html.charters/simple-charter.html`) and SIPPING (`http://www.softarmor.com/sipping/`)[1]. With the significant features of SIP, such as scalability, reusability, and interoperability, this protocol has stimulated research for the third generation mobile networks. The third generation partnership project (3GPP) [GM02] has selected SIP as its signaling protocol for call control.

The objective of this chapter is to study the existing trend and application areas of SIP in the field of Information and Technology. The emphasis is on a comparative study of the SIP event package against various existing event architectures.

The rest of this chapter is organized as follows: an overview of the basic features and functionalities involved in SIP are explained in section 7.2. The research areas related to SIP and the different working groups are described in section 7.3. The SIP event architecture is explained in detail and

[1]The functions of SIMPLE and SIPPING working groups are discussed in subsection 7.3.2

compared with various existing distributed event standards in section 7.4. The work of Sun Java APIs for SIP is portrayed in section 7.5. A study of different research articles in the application area of SIP events is presented in section 7.6. Finally, in section 7.7, the report concludes with an overall summary of the topics discussed in the previous sections.

7.2 Overview of SIP

This chapter presents the basic concepts of SIP. The steps for establishing a session and the different transaction methods involved are discussed. Also, the salient features of SIP are explained.

7.2.1 Session Initiation Protocol (SIP)

SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions over the IP network [RSC$^+$02]. It is well suited for the requirements of presence applications as it routes the request from any user on the network to the server that holds the registration state of the user. Thus, SIP networks may be used to establish global connectivity for presence subscriptions and notifications, enabling it to be a suitable protocol for ubiquitous computing.

7.2.2 Terminologies in SIP

This section presents a brief summary of the central SIP terminologies. The definitions are from RFC 2543 (obsolete) [HSSR99] and [Cam01].

Dialog A peer-to-peer SIP relationship between two user agents that persists for some time.

Session A collection of participants, and streams of media between them, for the purposes of communication. Two familiar examples of sessions are Internet telephone calls and multimedia conferences.

Client A network element that sends SIP requests and receives SIP responses. Clients interact directly with a human user or a terminal device. Proxies and user agent clients are clients.

Server A network element that receives requests in order to service them and sends back responses to those requests. Servers include proxies, user agent servers, redirect servers, and registrars.

User agent The initiator of a SIP request (user agent client) or a responder (user agent server) on behalf of an end user. A user agent represents an end system containing a user agent client (UAC) to generate requests and a user agent server (UAS) answering to the requests from UAC. A UAC is capable of generating a request based on some external stimulus (e.g. the user clicking a button or a signal on a PSTN line) and processing a response. A UAS is capable of receiving a request and generating a response based on user input, external stimulus, the result of a program execution, or some other mechanism.

Back to Back User Agent (B2BUA) A logical entity that receives a request and processes it as a user agent server. In order to determine how the request should be answered, it acts as a user agent client (UAC) and generates requests.

UAS Core A set of processing functions required at a UAS that resides above the transaction and transport layers.

Registrars Entities co-located with a proxy or a redirect server accepting registration requests from users.

Redirect server A user agent server that generates 3xx responses to the requests it receives, directing the client to contact an alternate set of URIs.

Proxy server An intermediate entity that receives requests from a client (acting as a Server) and forwards or re-initiates the request (acting as a Client) to other servers. A proxy server can be either stateful or stateless. When stateful, it remembers incoming requests and their associated outgoing requests, and co-ordinates the responses accordingly.

Location server Used by the SIP redirect or proxy server to obtain information about the called party's possible locations. This may come from the SIP server or other protocols (non-SIP) when externally located.

SIP trapezoid The arrangement of SIP proxy servers that act on behalf of user agent clients to establish connections between them.

SIP entities Any entities present in the functionality of SIP. Examples are user agent clients and servers, stateless and stateful proxies, and registrars.

SIP transaction Occurs between a client and a server and comprises all messages from the first request sent from the client to the server up to the final (non-1xx) response sent from the server to the client.

7.2.3 Steps Involved in Establishing a Session

To establish a session, an INVITE request is sent to SIP Agent 2 from SIP Agent 1. SIP Agent 2 returns an OK response. After it receives an acknowledgement from SIP Agent 1 the session is established. Similarly a BYE and an OK message is used to terminate the session.

Figure 7.1 shows an example of session establishment between two SIP user agents.

1. The user agent client (caller) sends an invite request to the callee with the address of the callee. The address is similar to a mailto URL. For example, if the address of the callee is `callee@hiit.fi`, then the corresponding SIP URL will be `sip:callee@hiit.fi`.

2. The request will be sent to the SIP proxy. The location of the callee at that moment will be found out by the SIP proxy. To do so, the proxy sends the invitation to multiple locations and finds the exact place of the user at the moment. The process of sending an invitation to multiple points for finding the user's current location is termed "forking".

3. Finally, the request arrives at the UAC (callee). The callee can either accept or reject the request. If the callee accepts the request a confirmation is sent to the caller. A session is then established after the callee receives an acknowledgement from the caller. Here, the SIP proxy is responsible for delivering messages and responses between the caller and callee.

Address binding is the method used to register a user's current location when he switches on his SIP client. This registration is used to find out the user's location at any given moment.

7.2.4 Methods and Response in a SIP Transaction

There are six methods defined in SIP for session establishment. The number of methods increases with extensions and need for new applications based on SIP functionality.

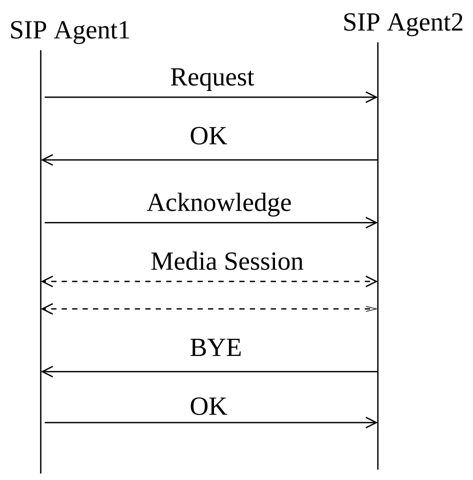

Figure 7.1: Session Establishment

INVITE (REQUEST) initiates a session, e.g. by inviting a user to join a call.

ACKNOWLEDGE (ACK) confirms that an INVITE has been received. ACK is used only as a response to INVITE.

BYE terminates the session.

OPTIONS queries a server about its capabilities.

CANCEL cancels a pending request, but does not end the session.

REGISTER registers with the location service.

The response codes given for SIP methods are described below and in [Sin01].

CODE	Description
1xx	Provisional or informational: Request is progressing but not yet complete
2xx	Success: Request has completed successfully
3xx	Redirection: Request should be tried at another location
4xx	Client Error: Request was not completed due to an error in the request; it can be retried when corrected
5xx	Server Error: Request was not completed due to an error in the recipient; it can be retried in another location
6xx	Global Failure: Request failed or should not be retried again

7.2.5 Features of SIP

SIP has many significant features, such as interoperability, scalability, and reusability. One of the important features considered throughout the design of SIP is interoperability. Thus, the characteristics of SIP enable application developers to add special features and extensions to the core concept of SIP.

SIP is designed to be scalable, and it can handle a large number of requests efficiently. To assure scalability, SIP servers are operated in two modes, stateless and stateful mode. Stateful proxies store, and stateless proxies do not store, the records of received and forwarded requests. Thus, stateless proxies are efficient in handling more requests, and can be used in the core network. Stateful proxies, which are well suited for the requirements of forking and multicasting are used at the edges of the network. This way of utilizing the proxy based on network demand makes SIP scalable in a distributed environment [Cam01]. The SIP server performance tests conducted by IPTEL (`http://www.iptel.org`), RADVISION (`http://www.radvision.com`), and Mocking bird networks (`http://www.mockingbird.com`) individually has shown efficient results.

Reusability is another feature of SIP. The protocol works well with many of the existing protocols that are used with HTTP. Hence, SIP matches the need without any requirements to build a totally new architecture. The addressing notations used by SIP are similar to that of HTTP. Also, SIP supports MIME and SMTP protocols for email transfer.

The main functionalities accomplished by SIP are

- Name translation and user location: call reaches the callee exactly at his current location.

- Feature negotiation: supported features are agreed on with the members involved in a group [BSSW03].
- Call participant management: management of actions that are taking place during a session, such as canceling, transferring, and holding calls.
- Call feature changes: changing call characteristics during a call, e.g. modifying the features.

7.3 SIP Research Areas

At present there are various research groups working on SIP and its applications. The IETF and its supplementary groups are involved in active research on SIP and its extensions. Other than the IETF, 3GPP is also designing the standardization for using SIP in mobile networks. The SIP Center (`http://www.sipcenter.com/`), formed by the co-operation of various industrial partners, and SIP Forum (`http://www.sipforum.org/`), a non-profit organization formed by individuals interested in SIP development with support from industrial partners, are the other active participants in the SIP community.

7.3.1 Active Research Topics

SIP is a prominent research area, since the SIP core concept and its extensions provide extensive applications in different fields ranging from telecommunication to e-commerce. Emerging research topics are

SIP mobility (Mobile SIP)

Mobile SIP studies the possibility of using SIP effectively for mobile communications. Here, mobility is defined with respect to terminal, personal, service, and session mobility [Cam01]. Henning Schulzrinne et al and Stefan Berger et al have discussed the application of SIP in mobile and ubiquitous computing [SW00, BSSW03]. Session mobility is not possible with Mobile IP. In Mobile IP a permanent IP address is necessary, and it is bound to a temporary care-of-address for identifying the host. Moreover, Mobile IP has the advantage of extending terminal mobility for a wide range of applications, and not constraining it to a very narrow area of telephony.

SIP, an application layer protocol, supports session mobility using mechanisms like third-party call control. SIP accepts the SIP identifier, which is similar to the format of an email address, and binds it to the temporary IP address or host name. Thus, the user is entitled to immediate access of mobile services. SIP has to use a separate anonymity service to hide the location details of the terminal. Comparing the advantages and disadvantages of Mobile IP and SIP, improved and additional applications can be formed by combining the salient features in both protocols.

Authentication, Authorization and Accounting (AAA) for SIP
The existing protocol RADIUS can be configured to fulfill the requirements of SIP [Cis03]. However, higher-level security features are required for AAA protocols as the complexity of the architecture increases with different applications of SIP.

Multicasting with SIP
The project MBone [SRL96], originally intended for researching an efficient multicasting system led to the development of SIP, a component of MBONE. Therefore, SIP was basically designed for enabling multicast sessions. While multicasting is an intensive area of study, research in mobile multicasting and related areas is in its infancy. The multicasting protocols designed for static environment are not suitable for the mobile environment, and hence a mobile-specific architecture is considered. The area of study in mobile multicasting includes quality of service, mobile host hand-off, and reliability. Christophe Jelger et al and Hrishikesh Gossain et al have analyzed the research issues involved in mobile multicasting [JN02, GdMCA02]. SIP can be applied to mobile multicasting as it supports mobility and has the features to handle multicast sessions.

SIP server performance
The basic issues that arise with quality of service should be managed. Network congestion and handover issues have to be studied in detail.

SIP for control applications
Control applications include call control services. Call control for telecommunication-based applications is an evolving field with the contribution of Sun's Java APIs such as JAIN and JCC. Also, research works are carried out studying the possibility of SIP call control in other communication areas, such as broadband satellite sys-

tems [Kar03]. SIP is considered to be a suitable protocol for home networking. Though there are many existing protocols for home networking, SIP gains higher importance as it has the required features of simplicity, scalability, and reusability.

Porting SIP to IPv6
Due to the unexpected growth of Internet usage, the IPv4 version of IP addressing is expected to get exhausted soon. IPv6, the proposed successive version, will be replacing IPv4. As a result all IP networks have to support IPv6 when the transition of IP addressing takes place. SIP, currently emerging as a signaling protocol for IP networks, should be capable of supporting IPv6. There are few successful implementations of SIP that support both versions of IP. Still, there are studies going on to ensure that SIP also works over IPv6.

Interoperability between SIMPLE and Jabber
Jabber (`http://www.jabber.org`) is an open source protocol for instant messaging based on the client-server architecture and implemented using XML. It is a tested protocol, and at present Jabber is used extensively by clients in instant messaging applications. SIMPLE, an extension of SIP supporting mobility and call control, could be interworked with Jabber to provide an efficient instant messaging system for both wired and wireless environments [Sau02].

7.3.2 Working Groups in IETF

There are three significant SIP-related working groups within the IETF:

Session Initiation Protocol Working Group (SIP WG)
The SIP WG concentrates on the specification of SIP and its extensions. The main target of this working group is to define the necessary end-to-end functionalities, generic features, and extensions of SIP. Effective usage of existing Internet protocols and architectures is one of the main factors considered to achieve the goal of this working group. The SIP working group co-ordinates with other IETF working groups, such as SIPPING, Multiparty Multimedia Session Control (MMUSIC), and PINT (PSTN and Internet Internetworking).

SIP for Instant Messaging and Presence Leveraging Extensions Working Group (SIMPLE WG)

The SIMPLE WG deals with the application of SIP in the services of Instant messaging, presence, and session-oriented applications. The extensions defined by SIMPLE should be compliant with the basic behavior of SIP. SIMPLE works in association with the Instant Messaging and Presence Protocol (IMPP) and SIP working groups.

Session Initiation Proposal InvestiGation Working Group (SIPPING WG)

The SIPPING WG studies the possible applications of SIP related to telephony and multimedia. Also, the group analyzes the extensions required to implement the applications. The specific task of the WG are standardizing a framework of SIP for telephony, application of AAA in SIP telephony, analyzing the multi-party applications of SIP, and SIP interaction with media servers. This working group works in co-operation with SIP, SIMPLE, MMUSIC, Authentication Authorization and Accounting (AAA), IP Telephony (IPTEL), and PINT working groups.

7.3.3 Work in the Third Generation Partnership Project on SIP

The 3GPP was formed in 1998. The objective of the 3GPP is to co-ordinate the activities of various telecommunication bodies such as ARIB[1], CCSA[2], T1[3], ETSI[4], TTC,[5] and TTA[6]. SIP has been selected as the session establishment protocol for the 3GPP IP Multimedia Core Network Subsystem (IMS) [GM02].

The 3GPP has identified certain requirements for SIP to work efficiently over cellular networks:

- The signaling protocol should efficiently use the radio interface.
- In a cellular network, as the size of terminals used are small, the protocols should require minimal power and memory.
- The time taken to initiate a session should be minimal.

[1] Association of Radio Industries and Business, Japan
[2] China Communications Standards Association
[3] Telecommunications, USA
[4] The European Telecommunication Standards Institute
[5] The Telecommunication Technology Committee, Japan
[6] Telecommunications Technology Association, Korea

- The signaling procedure should be similar for roaming and non-roaming situations.
- Mobility management of terminals is carried out by the access network, and hence there is no need for SIP to have terminal mobility management.
- IPv6 is used in the IMS, and therefore SIP is expected to support IPv6.

A SIP outbound proxy is used to support both roaming and non-roaming procedures in the network, and hence the mobile device (UA) needs to know the address of the SIP outbound proxy server. DHCPv6 for SIP servers helps the UA to locate the SIP outbound proxy server address.

Registration and Deregistration

The necessity and features of registration and deregistration procedures are explained clearly in the SIP event packages. When the SIP terminal is switched on, the configuration data containing the identity of the home network of the device is read by the UA and stored in the memory of a Subscriber Identity Module (SIM) card or some other memory device. This identity of the home network is used by the device to know the SIP registration address. The registration is forwarded by the terminal through the SIP outbound proxy to the SIP registration address.

The registration is essential because of the following reasons:

1. The UA has to register before sending or receiving a session request.
2. The SIP server proxy requires the details, such as when and where the user and terminal are located, from the information furnished in the registration.
3. During registration, pre-authentication is done to avoid post-dial delays.
4. Once registered, the user is assigned to a particular serving proxy which downloads the service profile for the user to trigger services.

A single registration should be sufficient to locate the user at both home and visited networks. The registration for roaming and non-roaming procedures should be the same. The process of deregistration, which is equally important, is executed by registering with the expiration timer set to zero.

The circumstances where network-initiated deregistration becomes unavoidable are

- The SIP server proxy needs to be shut down due to unexpected happenings in the network.
- When the user roams to a different network, deregistration is essential to avoid inconsistent information and redundant data storage.
- As the subscription timer expires, the subscribed events have to be canceled by the administrative function of the SIP proxy server. Also, to overcome the issues arising due to e.g. a stolen terminal, cancellation of the subscription is important.

SIP compression techniques are identified by 3GPP to transfer long SIP messages over the air interface and to reduce the session set up time. The other characteristics to be considered with SIP are quality of service, prevention of service theft, and prevention of malicious usage.

Identification of SIP Users over the Cellular Network

A given private user identity is unique for all global users, and it can not be used for routing SIP messages. This identity is permanently allocated to a user based on his business subscription with the home network. Private user identity can be effectively used for authentication, authorization, and administration as well as for accounting purposes.

One or more identities are allocated to a user to communicate with other public users. This public identity has to be registered with the SIP registrar before becoming involved in a session. The public user identity is authenticated by the network, and checked for its association with the correct private identity.

7.4 SIP and Event Architectures

Distributed event systems play a vital role in facilitating asynchronous information dissemination. As a newly designed event architecture, SIP utility has higher possibility of usage in different event-based applications. To become an ideal event system, the event standard has to prove that it has beneficial features compared to existing systems. Also, event systems for the mobile environment have to handle events efficiently with smaller memory size and less power consumption, and use less bandwidth. Thus, research into the SIP event system is challenging so as to make it an ideal event system which can be used in wired, wireless, and mobile environments.

7.4.1 An Introduction to Events in Distributed Systems

Events are expected occurrences when a predefined condition is met, or when some expected change occurs in the system. Event monitoring and notification services are required to monitor the events and relay them to the desired clients in a distributed environment. The application of event services in a mobile environment is an emerging field with increasing response from the end-users.The two main actors involved in an event service are

Consumer (Subscriber, Listener, Observer)
Interested in getting the notifications about events; subscribes for particular events.

Supplier (Producer, Notifier, Publisher, Subject)
Notifies a consumer when an event occurs to which that consumer has subscribed.

The processes involved in an event service are event subscription, monitoring, and notification. There are many existing event frameworks for distributed systems such as CORBA, JINI, and SOAP grid events. Each standard has its own merits and demerits. Recently, the event package of SIP is becoming more popular compared to the presently used event architectures in mobile applications. Any event framework should overcome the basic issues of security, interoperability, and extensibility. This chapter treats briefly the features of the existing event frameworks of CORBA Notification Service, JINI, and Grid SOAP events. Also, a comparative study of these event frameworks with that of the SIP event package is presented in detail.

7.4.2 General Requirements of an Event Framework

Various event frameworks exist for the distributed computing field. The functional requirements for an event framework can be classified as follows [SGGB02]:

- Filtering options: The subscription method for subscribing to the events should include a filter option to ensure that the notification is sent only for the subscribed events.
- Expressiveness and scalability of the filtering language.
- Interoperability: A main factor if the subscription and notification functions have to be performed across various platforms.

- Quality of service.

Mobility support and buffering are also essential functional requirements in a ubiquitous computing environment. The non-functional features required in an event system include fault tolerance, scalability, and security.

7.4.3 Design Patterns for Event Architectures

There are two well-known design patterns based on distributed event architecture, namely the observer and notifier patterns. The observer pattern is a simple model, with a subject and an observer. The subscriber registers to a specific event, and the subject notifies the subscriber when there is an update. In this pattern, the subject needs to know about the subscriber. Hence, this pattern becomes complicated in a wide distributed architecture where the producer has to keep track of all the consumers subscribed for each event.

The notifier pattern has an advantage of having a broker architecture that separates the subscriber and notifier. Event subscription in the notifier pattern is based on the events in which the subscriber is interested in instead of subscribing by knowing the details of who is publishing the events.

The observer pattern is used in the SIP and JINI event architectures whereas a variant of the notifier pattern is used in the CORBA Notification service. The observer pattern is known for its simple design, while notifier is an extended model of the observer pattern with the broker pattern.

7.4.4 SIP Event Notification

RFC 3265 has been framed for SIP-specific event notification. The event package enables the client/user-agent to subscribe to the desired events and get notified when an expected event occurs. The application of event packages can be seen in automatic call-back services, buddy lists (presence), and message waiting indications.

The terminologies involved in the event package are listed as follows. The definitions are taken from [Roa02].

Event Package An additional specification when an asynchronous event occurs, and gets notified to the subscribed client.

Notification An act of a notifier sending a NOTIFY message to a subscriber of the state.

Notifier A user agent that generates NOTIFY requests for the purpose of notifying subscribers of the states.

State Agent A notifier that publishes state information on behalf of a resource.

Subscriber A user agent that receives NOTIFY requests from notifiers containing information about the state of a resource in which the subscriber is interested.

Subscription A set of application state associated with a dialog.

Subscription migration The act of moving subscription from one notifier to the other.

The important methods dealing with event notification are Subscribe and Notify. Both methods are discussed in detail in the following part of this section.

SUBSCRIBE

This method is used to request current state and state updates from a remote node [Roa02]. The subscribe method contains the following entities:

Expiry time A subscription is valid up to the expiry time specified in the header. The subscription has to be refreshed before it expires, and if the expiry time is not explicitly mentioned, a default value defined by the event package is used.

Request URI This is used to route the subscribe request to the appropriate entity.

Event header This shows the subscribed event class. There should be exactly one event header specified and the event identity is optional. The identity is useful in differentiating between multiple subscriptions within the same dialog [Roa02].

A successful subscribe request receives a 200-class response. A 202 response shows that the subscription has been received but the subscriber has not been subscribed to the event yet. Non-200-class responses show failure to subscribe to the event, and hence no notification will be sent. Successful unsubscription from an event also receives a notification. If the subscription is not renewed before its expiry time the subscription will be terminated, and the information will be notified to the subscriber with a `reason=time-out` parameter in the subscription state header.

Figure 7.2 shows the steps involved in an event subscription and notification. The subscriber SUBSCRIBEs for an event and gets a 200 confirmation response. Once the subscribed event occurs the notifier sends a NOTIFY to the subscriber and receives a response as confirmation of success.

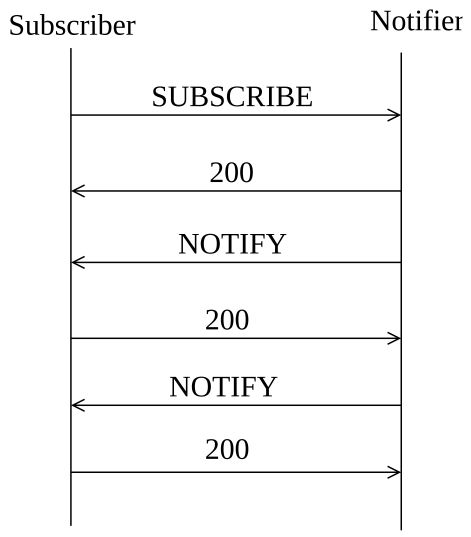

Figure 7.2: Event Subscription and Notification

NOTIFY

This message is constructed immediately after the 200-class response is sent to the subscribe request. It is used to notify the subscriber when a subscribed event occurs. The notify request fails if the response times out, or an unsuccessful-class response is sent.

The subscription state header in NOTIFY should indicate the status of the subscription. The value of the subscription state hence shows whether the subscription is still active or pending. Responses such as retry-after indicate that the subscription request has not yet been confirmed.

The various reason codes defined for the notify methods in RFC 3265 are listed as follows:

deactivated indicates the termination of the subscription and hence the subscriber can try sending a new subscription request immediately.

probation shows the termination of the subscription and the subscriber is expected to retry after the specified time, if the retry-after parameter is included in the header.

rejected states the rejection of the subscription due to authorization policies. The client should not retry.

timeout expresses the termination of the subscription as the time limit expires. The subscriber can immediately send a new subscription request.

giveup indicates the termination of the subscription due to problems in the time factor for authorization. The subscriber may retry after some time as indicated in the retry-after parameter.

noresource subscription terminated as no resource was detected to monitor the event.

Other than the subscription state value, the event header similar to that of the subscribe method should be available. Also, the event identifier is optional as in the case of subscribe method. The body of the notification method contains the state of the resource. If the state information is too big to transfer, a URI value is returned from where the state information can be accessed by the subscriber.

The Call Sequence (CSeq) is used to maintain the order of the transaction while the Call-ID uniquely identifies the particular invitation or registration from a client. The Call-ID can also be termed as an unique global identifier [RSC$^+$02].

7.4.5 CORBA Notification Service

The CORBA Notification Service [OMG01b] from the Object Management Group (OMG) is an accepted solution set of the 3GPP for developing telecommunication applications. The basic architecture follows the mixed push-pull model and the notifier/event-channel design pattern. According to the notifier design pattern, CORBA Notification Service uses an event channel that decouples the consumer and the producer. Based on the mixed push-pull model, the invocation on an object in the event channel is done by both the supplier and consumer to push and pull events respectively to and from the event channel. Moreover, CORBA Notification Service supports event filtering. Filtering based on event priority and lifetime can be implemented in addition to forward filtering that filters and forwards the required events as requested by the consumer [ION02]. CORBA Notification Service has been discussed in subsection 2.3.5.

7.4.6 JINI Event Architecture

The JINI event architecture basically uses the RMI (http://java.sun.com/products/jdk/rmi/) protocol and follows the listener pattern, a specific version of the observer pattern. The JINI specification requires an event identifier and a sequence number to uniquely differentiate the generated events with the same event identifier. To avoid loss of data, maintaining sequence numbers for events is essential in disconnected networks. The JINI event system allows the addition of an application-specific middleman architecture to perform filtering [Li00]. A detailed analysis of the Java distributed event model is given in subsection 2.3.2.

7.4.7 GRID SOAP Event Systems

This event system supports the additional features of event channels, leasing, and filtering. It follows many of the features that are present in the JINI event architecture. With GRID SOAP events the preferred features for smaller devices, such as limited memory usage and resources, can be achieved. The event publisher simply writes a pre-formatted set of strings into the transport layer. SOAP is a light-weight protocol, and it is being supported by many programming languages. Thus, GRID SOAP events are more suitable for distributed environment [SGGB02].

7.4.8 Comparison between SIP, CORBA, GRID SOAP, and JINI Events

Based on the various distributed event architectures analyzed so far, the event filtering techniques and mobility support of the event systems are compared in this section.

Event filtering is an open issue in the SIP event architecture. As the SIP event package supports extensible implementation, a filtering system could be included into it. XML and SOAP messages can be enveloped with the subscription and notification message bodies. The SIP event package supports subscription and notification life-time by adding an expiry time value in its method headers. SIP supports personal mobility using the forking technique. Also, SIP supports terminal, service, and session mobility.

The CORBA Notification Service provides a well-defined mechanism for event filtering and supports buffering. Life-time values are included in the property settings of filtering methods. Management of event domains is addressed well in CORBA Notification Service and has been accepted as a standard protocol for telecommunication applications by the 3GPP. An overview of CORBA domain event management is presented in subsection 2.3.6

The JINI event system uses a middleman mechanism for filtering. The lifetime of the subscription and notification methods in event systems are carried out using leasing techniques. The basic JINI specification does not provide any special mechanisms for mobility support. JINI is not suitable for the mobile environment as it follows the RMI protocol which is less efficient for mobile services [Cam03]. Also, JINI does not support service mobility. JINI events are being implemented and tested for home networking.

GRID SOAP events include filtering. The life-time properties of notification and subscription methods are set using a leasing technique. GRID SOAP events do support the buffering features which are useful in the mobile environment.

7.5 SIP and Java

This chapter gives a brief introduction to a Java-based implementation for SIP. The various SIP implementation standards such as JAIN, SIP Servlets, and J2ME are discussed.

7.5.1 JSIP — A Prototype Implementation of SIP Extensions

Session establishment between clients using the jSIP is performed as defined in RFC 2543 [HSSR99]. Also, it includes a prototype implementation of a few SIP extensions, e.g. instant messaging applications. The download and API manual of the jSIP are available at SourceForge (`http://jsip.sourceforge.net/`).

7.5.2 SIP and JAIN

Java for Integrated Networks (JAIN) is a community extension of Java technology developed by Sun's Java Specification Participation Agreement

(JSPA) and Java Community Process (JCP). JAIN enables the integration of the Internet and intelligent network protocols. JAIN is designed with the aim of co-ordinating the three network layers, namely the network, signaling, and service layers. The areas of focus in JAIN are

- Protocol API specifications: Specifies interfaces to wire-line, wireless, and IP signaling protocols.
- Application API specifications: Addresses the APIs required for service creation within a Java framework across all protocols covered by the protocol API specifications.

JAIN has also defined the Java Call Control (JCC) API for call control applications in telecommunications. JCC can be mapped to SIP where the SIP application is built over the JCC implementation [JBA01].

The JAIN SIP API includes user agent, proxy, and redirect server interfaces. JAIN SIP Lite is a high level Java API for application development for the J2SE and J2ME platforms. Developers need not worry about the underlying SIP protocol.

7.5.3 SIP for J2ME

SIP for J2ME is the Sun Java specification for light-weight and resource-constrained devices, such as mobile phones. The Java Specification Requirement (JSR) draft has been published for SIP J2ME. Though SIPLite can also be used in hand-held devices, such as Personal Data Assistants (PDA), J2ME is mostly considered to be an architecture for mobile devices.

7.5.4 SIP Servlets

SIP Servlets is a high-level Java API specifically for SIP servers in telecommunication-based applications. SIP servlets can create and access session, call, and transact data. Unlike HTTP servlets, SIP servlets interact with the SIP, application, and proxy servers.

7.5.5 Interworking of Various Java APIs with SIP

Since the line of difference is thin between the functions performed by the various Java SIP APIs, it is important to provide a clear differentiation. SIPLite and Servlets are aimed at developers without a background in the SIP protocol whereas JAIN needs expertise in both the protocol and Java.

JAIN is developed for J2SE, J2ME for mobile devices, Servlets for J2EE and SIPLite is for both J2EE and mobile devices. Figure 7.3 shows the IMS architecture designed by 3GPP integrating various Java APIs with SIP.

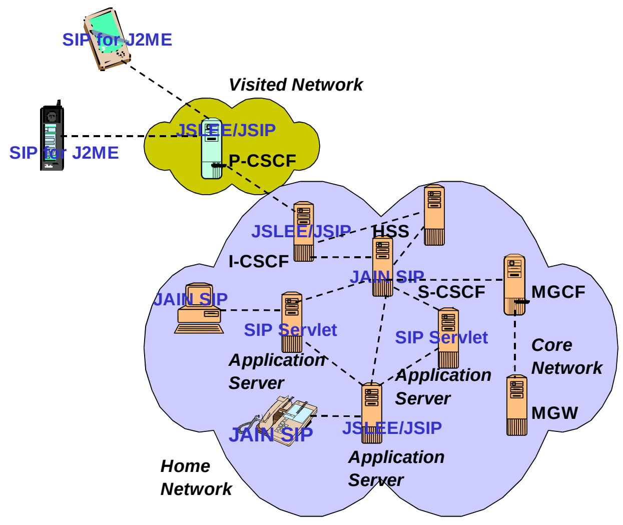

Figure 7.3: IMS architecture [O'D03]

7.5.6 Open Source SIP Implementations

There are a few open source implementations available online:

- The SIP developer's web page (`http://communications.dev.java.net/`) provides installation packages for SIP presence, a JAIN SIP proxy server, and a JAIN SIP proxy client.
- The siptrex (`http://www.siptrex.net/news/`) provides a SIP user agent and proxy for IP telephony services used by application service providers.
- The University of Columbia's Internet Real Time laboratory's research group (`http://www1.cs.columbia.edu/~pallavi/research/using.html`) working under signaling and control protocols has a SIP library

and example codes. Also, a SIP user agent is available for download at http://www1.cs.columbia.edu/~xiaotaow/sipc/index.html.

- The National Institute of Standards and Technology (NIST), provides the implementation of JAIN-SIP interfaces used for further application developments at https://jain-sip.dev.java.net/.

- In Vovida (https://www.vovida.org/) downloads are available for SIP Residential gateway and SIP User Agent.

- Asterisk (https://www.asterisk.org/) is a software PBX that runs on the Linux platform. The services given by Asterisk include voice mail services, directory services, and call conferencing.

7.6 Related Work

7.6.1 Medical Event-based Monitoring System

Medical event-based monitoring system [AS03] suggests the use of SIP, Bluetooth, and SOAP for the medical event-based monitoring system. According to the scenario explained, the heart monitor sends signals to the Bluetooth access point that is used to find out the doctor's current location. The access point is connected to a SIP server to which the hand-held device of the doctor has subscribed for the event. Thus, the doctor receives the update about the patient's health through the notification of a subscribed event to his hand-held device.

In this system, the SIP event methods Subscribe and Notify play a lead role. The author has suggested the use of SOAP for Remote Procedure Call with SIP. The feature of SOAP as an interoperable protocol in multiple domains is a key requirement in the medical field.

7.6.2 Application of SIP to Ubiquitous Computing

The architectural description of a service discovery system, location sensing, and call control using SIP, Bluetooth, and Service Location Protocol (SLP) is discussed in [BSSW03]. The information about the location of the user can be identified using the Session Description Protocol.

Location sensing includes the functionalities of determining user location and publishing location information. Based on the location information received, certain automated events are triggered and the communication behavior is controlled. Bluetooth is used to determine user location and SIP is used to publish the location information. The SIP methods REGISTER and PUBLISH are used for this purpose. Also, the event-triggered actions and communication control are handled by the SIP presencing and event packages.

Stefan Berger et al [BSSW03] have explored the possibilities of using SIP messaging, SOAP, and HTTP for implementing control messaging between terminal devices.

The service example is demonstrated with a SIP-based ubiquitous computing environment. The user entering into a video conferencing room with his wireless device is first authenticated and authorized using the access control technique. Once the user has successfully entered the video conference room, his credentials are added to the ubiquitous computing system. The user's mobile device gets the new IP address from the visited domain and informs the SIP server in the home domain of this new IP address using the REGISTER method. Here the REGISTER method also carries location information received from the Bluetooth devices. The SIP server in the home domain can now perform the call control operations for the user after getting the REGISTER request. Using SLP, the list of devices available at the current location and their capabilities are made available to the user's end system.

7.6.3 SIP for Emergency Systems

The Government office information systems subscribe to events in SIP-based emergency systems and, they are notified in case of emergency. The present method used for emergency systems is digital broadcasting. UAC is the entity that subscribes to the event and UAS is the entity that notifies the UAC of the occurrence of an event. Based on the information received in the message body of NOTIFY, the UAC produces an alert signal.

An authentication system is embedded in the architecture to check the validity of UAC. Knarig Arabshian et al suggest the usage of SOAP within the NOTIFY messaging system [AS01a].

7.6.4 SIP Extensions for Communicating with Networked Appliances

The IETF draft submitted by Hughes Software [TCH00] suggests the application of SIP in the networked appliances using the proposed 'DO' method, for example, in a home networking system where the consumer devices are networked with each other. The body of the 'DO' method carries the action to be performed by the networked appliance. An authentication system has to be included in the architecture.

7.7 Summary

The application of distributed event delivery plays a vital role in the emerging field of mobile applications. Providing an efficient event service is becoming a challenging task in the mobile environment. Current open issues, such as roaming clients with fast handovers and disconnected clients are yet to be answered for many existing event architectures. SIP events as a developing standard should be able to meet these requirements.

From the analysis performed on various existing event standards it is evident that there is no ideal event architecture which best suits the need for wired, wireless, mobile, and ubiquitous environments.

SIP events have certain advantages over the other event architectures. In a middleware event architecture such as CORBA, developing an event system includes issues related to session management. In the case of SIP events, which are an extension of SIP, session management is handled efficiently. Hence, when building applications based on the event architecture, the application developer need not give special attention to session related issues. Also, Sun Microsystems has designed the specification requirements for SIP services specific to various application environments in Java Community Process (JCP). Thus, SIP event package requirements can be sorted out easily with distinct Java APIs for different environments.

SIP events support mobility and session management. Still, they have to provide the essential properties required for mobile environment, namely filtering and buffering.

Chapter 8

Context Modeling

8.1 Introduction

The word “context” is built from the words “con” (with) and “text”, thus context refers to the meaning that can be inferred from adjacent text. In computer science, the most used definition of context comes from [Dey01]: “Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.”

Examples of contextual information include

- user identity
- spatial information (location, orientation, speed, acceleration)
- temporal information (time of day, date, season of the year)
- environmental information (temperature, air quality, light or noise level)
- social situation (who you are with, people that are nearby)
- resources that are nearby (accessible devices, networks, hosts)
- availability of resources (battery, display, network, bandwidth)
- physiological measurements (blood pressure, heart rate, respiration rate, muscle activity, tone of voice)
- activity (talking, reading, walking, running, sleeping)
- schedules and agendas

A system can be said to be context-aware if it makes use of contextual information. Humans are in general very good at using implicit situational information. For computers, interpreting and deducing context information are hard tasks with many problems. For instance, to perform the task “Reserve a table from a restaurant near to me”, it is required to have a precise definition of terms used in the task, particularly what “near” means to “me”.

Whereas the slogan of pervasive computing is to give access to information “anytime and anywhere,” for context-aware systems it is “say the ‘right’ thing at the ‘right’ time in the ‘right’ way” [FY01]. Key research challenges for context awareness include [CFJ04, EBDN03]

- Defining an explicit representation of context that is suitable for knowledge sharing, data fusion, and interpretation.
- Constructing reasoning mechanisms for deduction of contextual situation and detection, and resolving inconsistent contextual knowledge.
- Implementing an adequate framework for user privacy protection.
- Distributed maintenance of context information: extraction, storage, updating, deleting, modifying.
- Ways to express dependency and relevancy between pieces of context information.

Of these, our focus is on the first one: context representation that has a suitable level of formality and expressiveness for enabling context modeling and reasoning. For this report, we define context modeling to be the art of representing context information in a form that enables advanced context reasoning and maintenance. This means that we do not consider sensor fusion performed by probabilistic models, which could be called “context modeling” too; our interest is on information representation. For other machine learning methods used for context modeling, see [DSDE03] and [DSBE03].

The vocabulary of context-related research is quite unsettled. For this report we use the following definitions:

- The datasets where context information is stored are called context profiles.
- The distributed maintenance of these profiles is called context maintenance.

8.2 Requirements for Context Representation

Requirements for context profile representation used in a mobile environment include [HBS02]

- Interchangeability: Context profiles must be interchangeable between different system components and parties.

- Composability/decomposability: Because it must be possible to maintain context information in a distributed way, it must be possible to compose and decompose profiles. This enables, for example, sending of only the changed part of a given profile. Decomposability is also required in order to be able to handle privacy, i.e. separating possibly private information from a profile before sharing it.

- Extensibility: The number of possible context variables is, even in quite restricted domains, virtually unlimited. Thus extensibility of the representation by introducing new terms, context variables, is an important issue.

- Expression of dependability: Context information can be interpreted from other bits of context information, and thus depend on them; if the value of some piece of information changes, so should the value of the dependent context information. There must be a way to express this dependency, and dependency between different profiles must be supported.

- Shared understanding: In an open environment, where basically anyone can contribute to the common pool of knowledge, shared understanding between different parties is required.

Possible additional beneficial requirements, which make the processing of context profiles more convenient, include

- Formality: Context interpretation needs a formal representation with sufficient expressiveness. The amount of expressiveness needed to support context reasoning is still an open question and must be investigated further, as resources are scarce in a mobile setting. Formality is also one of the keys for shared understanding.

- Uniformity: In order to ease interpretation and deduction, uniformity of the representation in profiles is desirable.

- Standardization: The best way to ensure shared understanding, uniformity, and interoperability is standardization of the representation format of profiles.

8.3 Classification of Context Information

In this section, we present different classifications of context, the focus being on the mobile environment. They serve as a historical introduction to more recent models. Schirmer et al [SB00] came to the following conclusions when analyzing context: The traditional notion of context for computer programs was formed of the user running the program, and the computing resources and information used by the program. By adding the location of the user to the context, the attributes of the environment are taken into account. These attributes are grouped into three categories, namely the physical, technical, and social spheres. They also classified context along 3 axes:

- Change of value
  - Static: hardly changing, e.g. resolution of a display
  - Dynamic: changing depending on time and location, e.g. user activity
- Scope
  - Local: “micro world”, associated with a single object, e.g. resolution of a display
  - Global: “macro world”, correlating multiple objects, e.g. location of a user
- Interpretation
  - System-oriented: unequivocal characteristics of system components
  - Application-oriented: dependent on interpretation with different uses

Another way to classify context information is to divide it into layers. For example, in [AAH+02] the following classification has been used:

- On the lowest level, the physical layer, there are sensors and other objects, producing output in a raw format. Examples of these are an analogue microphone signal and the strength of a RF signal sent by a WLAN access point.
- On the second level, the data layer, there are objects producing processed data, for example spectral information of the phonemes in the audio signal or location coordinates computed based on three RF signal strengths.
- The third level, the semantic layer, contains objects which transform the data into a form meaningful for inferring context.
- The fourth level, the inference layer, uses information from the semantic level, earlier information and inference rules, possibly dynamically learned, to make educated guesses what the user (either man or machine) is doing and what kind of services he, she, or it might want.
- On the uppermost level, the application layer, applications or agents make decisions on behalf of the user by using the inferred information.

The objects on higher layers may combine input from one or more objects on the lower layers with stored information.

8.4 Ontologies

Ontologies play a central role in context modeling, especially the generic context modeling that we are after, i.e. the representation of contextual information in a way that enables interoperability via shared understanding of information and also allows easy introduction of new terms when needed. Also, formality that enables deduction is an important area where ontologies have a long history of research.

An ontology is a specification of a conceptualization [Gru93]. In other words, an ontology is an explicit formal specification of terms and relations between terms in a domain. This enables sharing and reuse of knowledge. While the best-known ontologies are huge, e.g. WebOnto that is being created for the Semantic Web, not all ontologies are a result of large-scale collaborative efforts trying to cope with “everything under the Sun”. For example, standardization efforts at expressing device capabilities, such as CC/PP, formally can be said to be ontological in nature.

Ontologies play an important role in distributed systems, because they may be stored at different places and created by different authors, which offers flexibility and extensibility. While many issues in collaborative maintenance of ontologies are still unsolved, these problems are at least well understood in this particular research area — in e.g. context modeling, the same problems seem to have been quite overlooked until just recently. One example of the maintenance problem is defining a new term to an ontology.

In order to create computer systems that can "understand" and make full use of a context model, the contextual information must be explicitly represented so that it can be processed and reasoned about by computer systems. Furthermore, shared ontologies enable independently developed context-aware systems to share their knowledge and beliefs about context, reducing the cost and redundancy in context sensing [CFJ03].

The reason for using ontologies as fundamental construction pieces for context representation is not only theoretically justified, but practical: the field has produced a vast number of different kinds of tools for handling the creation, evolution, and merging of ontologies. Many of the Semantic Web tools are in fact ontological tools. The field is also moving forward very fast, partially thanks to the hype around the Semantic Web, but also due to the maturing of the research field [McG02].

The merging of different ontology fragments is one of the main tasks of a reasoner, which is called an inference engine if it infers knowledge from symbolically coded axioms. A reasoner may be queried via some query language to deliver instances and their values, as well as concept and attribute names based on the ontologies known to the reasoner. A reasoner may also be used to validate consistency (within one ontology, but also with respect to related ones), and to assert inter-ontology relationships, and "complete" the ontologies by computing implicit hierarchies and relationships based on given rules.

In context modeling, ontologies can be used e.g. to describe users' task environments, as well as their goals, to enable reasoning about a user's needs and thus dynamically adapt to changes. Furthermore, descriptions of device capabilities and the devices' appropriate use will allow applications to reason about how to best support users in any given context [BB02].

Ontologies can be roughly divided into the following classes [FIP01]:

- Top-level ontologies describe very general concepts like space, time, matter, object, event, action, etc., which are independent of a particular problem or domain.

- Domain ontologies and task ontologies describe, respectively, the vocabulary related to a generic domain (like medicine, or automobiles) or a generic task or activity (like diagnosing or selling), by specializing the terms introduced in the top-level ontology.

- Application ontologies describe concepts depending both on a particular domain and task, which are often specializations of both of the related ontologies. These concepts often correspond to roles played by domain entities while performing a certain activity, like a replaceable unit or a spare component.

In practice, however, there does not exist a single commonly accepted definition for different types of ontologies.

In the following, we will briefly review, from the context-modeling point of view, different standardization efforts concerning some domain ontologies in the Fuego Core research area, and also the meta-ontology language, “language for defining ontology languages”, OWL [W3C03b].

8.4.1 CC/PP

Composite Capability/Preference Profile (CC/PP) describes an interoperable encoding for capabilities and preferences of user agents, specifically web browsers. Thus CC/PP can be seen as a domain ontology for user preferences and device capabilities. Because of its wide acceptance as a standard on field, it is important to support CC/PP for importing context information to the system.

While CC/PP is based on RDF, its Achilles’ heel is its poor extensibility resulting from a strict two-level hierarchy. In addition, attribute names are required to be unambiguous even if they are used in different components.

8.4.2 Device Independence Working Group

W3C’s Device Independence activity has drafted a classification trying to capture the general concepts of device independence from the web authoring point of view. Entities like “Web page”, “Functional user experience”, and “Delivery context” are defined with explanations, and examples are given. Defined concepts are used in stating “Device Independence Principles”.

In a way the current state of the work can be considered equal to the conceptual classification models mentioned earlier. In order to use the concepts for context information interpretation and deduction, the relations between concepts should be formally defined.

8.4.3 RDF

The Resource Description Framework (RDF) defines an XML language for expressing entity-relationship diagrams [W3C99c]. RDF defines standard tags for expressing a network of related objects. However, RDF does not specify a single logical model of entities or relationships: the same relationship could be encoded in many ways. XML and RDF are necessary but not sufficient for the exchange of complex information in open systems. One or more standard logical models are required in addition to constrain the use and interpretation of tags.

8.4.4 OWL Web Ontology Language

OWL Web Ontology Language is a language for defining and instantiating Web ontologies [W3C03a]. A definite advantage of having a standardized meta-ontology language such as OWL is that ontologies formed by it can use the same generic tools for reasoning. If a system was built with a specific industry-standard XML schema, tools specific to the particular subject domain would need to be created.

OWL (and to some extent also RDF) may be used as a metalanguage to define other special purpose languages, such as communication languages for knowledge sharing and policy languages for privacy and security. By defining the ontologies using OWL interoperability between different system components and systems can be increased.

An OWL ontology may include descriptions of classes, properties, and their instances. OWL has the most needed properties for ontologies, such as *owl:subClassOf*, which allows structuring entities to sub-class hierarchies, thus enabling introduction of new concepts to ontologies.

OWL comes in three flavors [W3C03a], OWL Lite, OWL DL, and OWL Full:

- OWL Lite supports users primarily needing a classification hierarchy and simple constraint features.

- OWL DL supports users who want maximal expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems. OWL DL includes all the OWL language constructs with restrictions such as type separation (a class can not also be an individual or property, a property can not also be an individual or class). OWL DL is so named due to its correspondence with description logics, a field of research that has studied a particular decidable fragment of first order logic. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems.

- OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

Because reasoners for OWL Lite are said to "have desirable computational properties", and OWL DL is subject to higher worst-case complexity, OWL Lite is a natural first candidate of the three for use in a mobile environment, especially if the reasoning or part of it is supposed to be done on resource-scarce devices. On the other hand, "saying the same thing" can sometimes be done in a shorter way with a more expressive language, so the choice might not be that simple if the computation can be done on the network side.

After an ontology has been created with OWL, the OWL formal semantics [W3C03c] specify how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms [W3C03a].

OWL has good expressive power for defining complex ontologies, and it has standard language syntaxes (e.g., XML, N3, N-Triple) for computer programs to process and manipulate the represented information [CFJ03]. The fact that OWL has good tool support that will be extensively broadened in the coming years lays a strong bias for using it as the language of choice for context reasoning.

OWL has already been used for creation of many special purpose languages, for example, an agent communication language [ZFD+03] and a security policy language [KFJ03].

8.5 Formal Approaches to Context Representation

In this section, we review the more formal approaches to context representation that enable reasoning. Also, context maintenance is reviewed via architectural description in cases where available. Coupling of architecture, representation, and reasoning is quite typical for the field: a clear sign of divergence and immaturity of the research area is the fact that the vast majority of research groups have built their own context-aware system from scratch, and research on context representation, maintenance, and reasoning is done solely on that system. Because of this, the used vocabulary and/or the meaning of terms also varies greatly between different research groups.

The focus is on research where the formalism used for context representation is provably expressive enough that it allows reasoning. Brief architectural descriptions are given when applicable in order to be able to evaluate the independence of the formalism from the architecture and get some understanding of context maintenance in the reviewed systems. This also helps the reader to understand the terms used in each system.

8.5.1 Comprehensive Structured Context Profiles (CSCP)

Held et al. [HBS02] describe a novel representation format that they call "Comprehensive Structured Context Profiles (CSCP)". While their focus is on content adaptation and they use only static and local context information, which does not go beyond the semantic layer, the format is interesting, as it is claimed to overcome the drawbacks of CC/PP mentioned earlier.

CSCP is an RDF-based meta language. As a descendant of RDF, CSCP inherits the interchangeability, decomposability, and extensibility of RDF. CSCP interchangeability is based on the XML serialization syntax of RDF.

Unlike CC/PP, the CSCP language does not define any fixed hierarchy. Rather, it supports the full flexibility of RDF to express natural structures of profile information. Attribute names are interpreted context-sensitively according to their position in the profile structure. Hence, unambiguous attribute naming across the whole profile (as necessary with CC/PP) is not required.

CSCP supports decomposability by means of external references and defaults. External references are used to extract sub-profiles to separate CSCP documents.

8.5.2 Model for Mobile User Context

One step forward from merely classifying context information is to build a conceptual model of it for a certain problem domain. In an attempt to model mobile user context, Tazari et al. [TGF03] identify the following groups of context data:

- Profiles of resources related to the user context, for example, available devices, services, documents, etc. Each such profile describes the identity, characteristics, and capabilities of the underlying resource and "knows" the location and state of the resource.
- Profiles of locations, which describe the identity and state of the location, and list the available resources and the people present at that location. The state of a location results from the perception of the physical characteristics of the location using sensor data, e.g. temperature, brightness, etc.
- The current time in diverse forms, e.g. absolute time, hour, AM/PM, etc.
- User profiles consisting of the user's identity, characteristics, capabilities, universal preferences, and the state of the user. A user's state includes information about his or her main activity, current terminal, etc., and a reference to a location profile.
- Application-specific user preferences.
- Other application-specific data that may play a role in the process of determining the user context, especially applications from the domain of personal information management (PIM), e.g. calendar, to-do list, address book, etc.

After this, they differentiate some classes of information from the rest on the basis that these could be modeled more or less independently from the applications. A model based on the general relations between the classes is presented. They utilize existing ontologies heavily for information representation within each class:

- Documents: PRISM, which is an enhancement of Dublin Core
- Services: DAML-S
- Terminal profiles: UAProf

The modeling of locations and users in their approach is not considered here further. Interestingly, they model user tasks with a hierarchical refinement approach with relations "parent" and "subtask". Tasks can also have "after" and "before" relations with each other, making one task marked as a prerequisite of another. While these relations are sensible and should be taken into account somehow when modeling user tasks, it is questionable how well a relatively static model based on these relations can be created and maintained, or how these relations are going to be used if they are inherently dynamic.

8.5.3 ASC-Model and Context Ontology Language (CoOL)

Strang et al. [SLPF03] describe a context modeling approach using ontologies as a formal fundament. They argue that approaches from the early days of context modeling usually lack formality and are primarily concerned with requirements for the model from the customer perspective. They introduce an "Aspect-Scale-Context (ASC) model" to close this formality gap. The context information is divided into concepts and facts, the relation of these being analogous to that of classes and their instances. The Aspect-Scale-Context (ASC) model is named after the core concepts of the model, which are aspect, scale, and context information. Each aspect aggregates one or more scales, and each scale aggregates one or more pieces of context information:

> An entity is a person, a place or in general an object. An aspect is a classification, symbol- or value-range, whose subsets are a superset of all reachable states, grouped in one or more related dimensions called scales. A context is the set of all context information characterizing the entities relevant for a specific task in their relevant aspects. An entity is relevant for a specific task, if its state is characterized at least concerning one relevant aspect. An aspect is relevant, if the state with respect to this aspect is accessed during a specific task or the state has any kind of in influence on the task. A system is context-aware, if it uses any kind of context information before or during service provisioning. The situation is the set of all known context information.

While similar to many other definitions of context, Strang et al try to overcome the problem of describing contextual facts and interrelationships in a precise and traceable manner by introducing the term aspect. Ontologies are used as a fundament to describe contextual facts and interrelationships, which helps to reach a degree of formality adequate for automatic interpretation capabilities of an implementation of the model.

A Context Ontology Language (CoOL) is derived from the ASC model. An important conclusion that Strang et al have made based on an analysis of languages that are built for describing ontologies is that these languages have a common trade-off between knowledge representation and querying. For this reason they define CoOL as a collection of fragments grouped into two subsets. The first is CoOL Core, a projection of the ASC model into two different ontology languages, namely OWL and F-Logic. The second is CoOL Integration, a collection of schema and protocol extensions as well as common sub-concepts of the ASC model.

F-Logic is a logic language combining object-oriented and predicate logic characteristics. Being able to use multiple ontology languages helps developers to choose the best tools for different tasks. While F-Logic is not based on XML, it is more expressive than OWL and argued to be better for specifying relevance conditions. Worthy of notice is that these definitions mean that relevance is considered to be more than just spatial and/or temporal proximity. An entity is considered relevant for a specific task if its state is characterized at least concerning one relevant aspect. An aspect is considered to be relevant if the state with respect to this aspect is accessed during a specific task or the state has any kind of influence on the task.

Besides new enhancements to formalism and general approach, what makes the ASC model and CoOL interesting is that the creators are able to show how the ASC model can be used as a transfer model, using DAML-S as an example. Further they show how the ASC model fits into a general-purpose service model, namely the MNM Service Model from the Munich Network Management team, and propose a system architecture that is able to affect service interaction at any stage.

While the focus of the ASC Model and CoOL is on allowing service interoperability on the context level, the approach seems to be promising and worth further investigation also from general context information representation and querying points of view.

8.5.4 Context Modeling via Dynamic Context Discovery

In a preliminary work, Thomson et al. [TRTN03] treat the problem of discovering and composing appropriate context entities as a special case of the more general problem of discovering and composing components in software engineering, and thus adapt previous research in software reuse to dynamic context discovery.

The infrastructure from their work is called the Strathclyde Context Infrastructure (SCI). It is organized into two distinct layers. The upper layer of the infrastructure is a network overlay of partially connected nodes and is referred to as the SCINET. The lower layer of the infrastructure concerns the contents of each node, which consists of entities (People, Software, Places, Devices, and Artifacts) responsible for producing, managing, and using contextual information, and is referred to as a Range.

The central concept in the approach is Range, which means a bounded physical or logical area. Range is occupied by one Context Server, which manages the other components and provides the means of communicating with other Ranges in a network.

Currently in the SCI, context information is represented as a configuration of context entities. The goal is to express context information on an abstract level, in terms of basic context elements and context operators, instead of concrete context entities and configurations. Basic context elements would be abstractions of context entities, and context operators structures for composition of basic context elements to build higher-level contexts.

To facilitate queries of this form, the context server of the Range is replaced with a semantically enhanced context trader. The context trader performs similarly to current trading services in that it takes a request for context information and returns a list of possible configurations sorted by some user-specified properties. However, the context trader differs from current trading services in that context entities include a behavioral specification as a part of their type description, and that matching of suitable entities is based on specification matching techniques.

The approach is interesting, as it would couple tightly the resources (services) producing context information to the formal representation of context, and methods from service discovery and composition areas might be used for context acquisition, etc. However, the feasibility of the approach is still an open question.

If the approach proves fruitful, it might easily be combined with e.g. the ASC model by substituting Range with the ASC model's definition of Relevancy.

8.5.5 GAIA

GAIA is an infrastructure for Smart Spaces by the Department of Computer Science at the University of Illinois [RMCM03, RHC+02]. Smart Spaces are "ubiquitous computing environments that encompass physical spaces". GAIA is a meta-operating system that aims to support the development and execution of portable applications for active spaces. Physical spaces and the computing devices they contain are in essence converted into a programmable computing system. This is done by offering services to manage and program a space and its associated state.

GAIA seems to be among the first, if not the first, context infrastructure to make extensive use of ontologies for context reasoning.

Architecture

GAIA is similar to traditional operating systems in that it manages the tasks common to all applications built for physical spaces. Each space can work independently, but may interact with other spaces.

> The main contribution of Gaia is not in the individual services, but instead, in the interaction of these services. This interaction allows users and developers to abstract ubiquitous computing environments as a single reactive and programmable entity instead of a collection of heterogeneous individual devices.

Three main components of GAIA are the Gaia Kernel, the Gaia Application Framework, and the Applications. The five basic services are:

- The Event Manager Service uses a decoupled communication model based on suppliers, consumers, and channels. It supports push, pull, and hybrid mechanisms. Furthermore, it supports the creation of named event channels and distribution of load.
- The Presence Service detects digital and physical entities that are present in an active space.
- The Context Service allows applications to query and subscribe for particular context information they are interested about.
- The Space Repository Service stores information on all software and hardware entities contained in the space, and allows browsing and retrieving them.
- The Context File System incorporates context into the traditional file system model to provide support for mobile users, device heterogeneity, and data organization.

The infrastructure is started by using a bootstrap protocol to start the basic services. GAIA uses CORBA to enable distributed entities to communicate with one another, but also customized solutions in order to handle e.g. crashing components and maintaining resources.

Ontologies are used extensively in GAIA to achieve better interoperability, not only in expressing context information, but also for different kinds of applications, services, devices, data sources, and other kinds of entities and the relations between them. Furthermore, axioms that must always be satisfied on the properties of said entities can be established. The use of ontologies is integrated via the Ontology Server, which maintains all the ontologies and allows getting descriptions of entities in the environment, meta-information about context, or definitions of terms used in GAIA. Support for semantic queries, e.g. classification of individuals, is planned in the future.

GAIA has several components dedicated to context maintenance. Context Providers obtain context from sensors, other Context Providers, or other data sources. Context Consumers, e.g. context-aware applications, can query them for context information. Context Providers can also have an event channel where they send context events. Context Synthesizers are used to derive higher-level context information from sensed context information and to provide inferred contexts to applications. Context Provider Lookup Service can be used by Context Providers to advertise the context they provide and by Context Consumers to find appropriate Context Providers.

Context Representation

Context is represented as predicates in GAIA, the structure of the context predicate depending on the type of context. Ontologies are used to define the vocabulary and types of arguments that may be used in the predicates, so that different entities have a common understanding of context. Ontology can also be used for checking the validity of context information.

Ontologies make it also easier for humans to specify how different applications and services should behave in different contexts, as the types of contexts that are available and their structure is known.

> There are different types of contexts that can be used by applications. These include physical contexts (like location, time), environmental contexts (weather, light and sound levels), informational contexts (stock quotes, sports scores), personal contexts (health, mood, schedule, activity), social contexts (group

activity, social relationships, whom one is in a room with), application contexts (email, web-sites visited) and system contexts (network traffic, status of printers).

Context predicates are expressed using a convention where the name of the predicate is the type of context that is being described, for example *Location(Bob, in, room 603)*. The structure of the context predicate depends on the type of context and this structure is defined in the ontology. For example, location context information must have three fields: a subject (person or object), a verb or a preposition like ‘entering’, ‘leaving’, or ‘in’, and a location like a room or an address. Each type of context is defined by a class in the ontology, for example the ‘Temperature’ context is a subclass of the more generic ‘WeatherInformation’ context.

Predicate arguments can be arbitrarily complex structures, as the context model makes no restriction on the types of values that different arguments in the context predicate can take. It is up to individual entities to interpret the location predicate appropriately. The use of ontologies helps ensure that different entities will interpret the predicate in the same way.

Boolean operators can be used to form more complex context expressions, for example: *EnvironmentLighting(Room 3234, Off) OR EnvironmentLighting(Room 603, Dim)* refers to the context that the lighting in Room 603 is either off or dim.

The model allows the use of universal and existential quantification over variables. For example, expressing that Bob is in some location can be written: $\exists_{Location} y \ Location(Bob, In, y)$.

In order to make calculations decidable, the model is based on many-sorted logic, where quantification is performed only over a specific domain of values. Because quantification is performed only over finite sets, evaluation of expressions with quantifications will always terminate.

One or more arguments of a context predicate can be a function returning some value. These functions are written in the C language. The presented representation formalism for context “can be mapped to any representation format like plain text, XML, a tuple in a relational database, or a serialized object”. In current implementation, the context predicate is mapped to a serialized object which contains the first order expression in a tree form.

Context Reasoning

The presented formalism based on first-order predicate logic allows the derivation of new contexts from other contexts. As an example of using rules to deduce new contexts based on existing context and to perform sensor fusion in a generic way, the following example is given: *Sound (Room 603, ">", 40 dB) AND Lighting (Room 603, Stroboscopic) AND nmbOfPeople (Room 603, " >", 6) ⇒ Social Activity (Room 603, Party)* This rule can be read as "a party is going on in a room if the level of sound in the room is high, stroboscopic lights are on, and the number of people in the room is greater than six."

Context information can be aggregated. Aggregate context of a smart space is the set of expressions involving context predicates that are true in that particular smart space. Context expressions can be generated by different Context Providers.

There are two basic protocols for obtaining context information: the query-answer protocol and the subscribe-notify protocol. In the query-answer protocol the query is given in a form similar to Prolog queries. In the subscribe-notify protocol a requester subscribes to certain contexts and gets notified whenever any of those contexts becomes true.

Applications can use context information in GAIA via rules that describe what actions should be taken in different contexts, for example : *IF Location(Bob, entering, workplace) AND Time(morning) THEN turn computer on.* The ontologies make writing these kinds of simple rules easy via a UI created for the purpose.

8.5.6 Context Broker Architecture (CoBrA)

Context Broker Architecture (CoBrA) is an architecture for supporting context-aware systems in smart spaces. It explores the use of Semantic Web languages for defining and publishing a context ontology, for sharing information about a context, and for reasoning over such information [CFJ03].

Architecture

A context broker has the following components:

- The Context Knowledge Base is a persistent storage of the context knowledge. It contains a set of ontologies for agents to describe contextual information and to share context knowledge, and it provides a set of APIs for other components in a broker to access the stored knowledge. Additionally it may contain heuristic knowledge associated with the space (e.g., a company's daily operation hours are between 9:00 AM to 5:00 PM).

- The Context Reasoning Engine is a reactive inference engine that reasons over the stored context knowledge.
- The Context Acquisition Module is a library of procedures that forms a middleware abstraction for context acquisition.
- The Policy Management Module is a set of inference rules that deduce instructions for enforcing user policies.

The Context Reasoning Engine can cope with two types of inference:

- Inference that uses ontologies to deduce context knowledge.
- Inference that uses heuristic knowledge to detect and resolve inconsistent knowledge.

The role of the Context Acquisition Module is similar to the role of the Context Widgets in the Context Toolkit [SDA99], which is to shield the low-level sensing implementations from the high-level applications. This module seems to be incorporated to the system architecture very recently as it is mentioned only in the latest paper.

> CoBrA maintains a model of the current context that can be shared by all devices, services and agents in the *same smart space*. The shared model is a repository of knowledge that describes the context associated with an environment.

In the Policy Management Module a policy language is used to define the permissions for different computing entities to share a particular piece of context information, and rules for selecting the recipients to receive notifications of context changes. Thus the users can protect their privacy by granting or denying the system permission to use or share their contextual information.

Context Representation

The ontologies of context are defined by using RDF and OWL, because “they provide an explicit semantic representation of context that is suitable for reasoning and knowledge sharing”. Based on these ontologies, the context brokers “can infer context knowledge (e.g. user intentions, roles, and duties) that cannot be easily acquired from the physical sensors, and can detect and resolve inconsistent knowledge that often occurs as a result of imperfect sensing”.

Instead of incorporating external ontologies, terms and organizations are adopted from existing ontologies such as the DAMLTime/Time-Entry ontology [Hob02], the OpenCyc spatial ontologies, the Friends-Of-A-Friend (FOAF) ontology, and the FIPA device ontology. This is justified as importing a substantial amount of irrelevant ontologies would hinder the performance of ontology reasoning. The plan is to rely on the OWL ontology mapping mechanism ([W3C03a]) in the future to support reasoning with the foreign ontologies.

At the current state of the system (which authors nominate as “preliminary”), all computing entities in a smart space are presumed to have a priori knowledge about the presence of a context broker, and the high-level agents are presumed to communicate with the broker using the standard FIPA Agent Communication Language [FIP02].

Context Reasoning

CoBrA seems to have relatively good support for reasoning in the form of different kinds of ontologies for its test-bed problem domain, intelligent meeting room. In addition to ontology inference, Context Brokers can also use logic inference to reason about contextual information. The Context Reasoning Engine has a two-tier design. At present, the ontology inferences on Tier 1 are limited to the OWL Lite subset of the OWL language.

As OWL does not support reasoning with uncertainty Tier 2 is for more complex reasoning outside OWL expressiveness. Currently it has customized rules for temporal reasoning and rules for interpreting the location context of a person and the status of a meeting. By using the device ontology, the physical location ontology, and the temporal ontology together, quite complicated and varied examples have been achieved.

8.5.7 SOCAM and CONON

Wang, Zhang, Gu and Pung have recently proposed [GWPZ04, WGZP04] a Service-Oriented Context-Aware Middleware (SOCAM) and Context Ontology (CONON).

Architecture

SOCAM aims to help application programmers build context-aware services more efficiently. It has the following elements:

- Context Providers abstract contexts from different sources and convert them to OWL representation so that contexts can be shared and reused by other SOCAM components.
- The Context Interpreter consists of Context Reasoning Engines and the Context KB (Knowledge Base).
- Context-aware Services make use of different levels of contexts and adapt the way they behave according to the current context.
- The Service Locating Service provides a mechanism where the Context Providers and the Context Interpreter can advertise their presences.

The Context Reasoning Engines provide context reasoning services including inferring deduced contexts, resolving context conflicts, and maintaining the consistency of the Context KB. Other components can query, add, delete, or modify context knowledge stored in the Context Database by using the service of the Knowledge Base.

SOCAM is said to be in prototype stage currently. The aim is to realize a scenario of a context-aware home, where various computing devices and physical sensors are present. The home network is connected to the Internet and the Context Interpreter is running on the OSGi gateway and implemented based on the semantic web toolkit Jena2. Thus SOCAM seems to be targeted for reasonably light weight use, and scalability is not very much an issue.

Context Representation

Perhaps a more interesting and unique part of the system is the Context Ontology CONON. Like for the majority of reviewed context ontologies, OWL is used also for CONON. While the authors acknowledge that “completely formalizing all context information is likely to be an insurmountable task”, they still believe however, that location, user, activity, and computational entity are “the most fundamental context for capturing the information about the executing situation”. They form the skeleton of context ontology and also act as indices into associated contextual information.

CONON is divided into an upper ontology and a set of domain specific ontologies. The low-level ontology in each subdomain can be dynamically plugged into and unplugged from the upper ontology when the environment is changed, for example between home, car, and work environments.

In the CONON upper ontology, the class *ContextEntity* provides an entry point of reference to the upper ontology: each user, agent, or service has their own instance of it. It has the subclasses *Person, Location, CompEntity*, and *Activity*. The refinement of the concepts of upper ontology is done in domain-specific ontologies.

In order to facilitate reasoning about the reliability of information, context information is classified into different types based on how its value is formed:

- *Sensed* context information gets its value based on physical or virtual sensors, e.g. sensor reading of a GPS device or web service call to a GSM location service.
- *Defined* context information is defined e.g. by a user or a device manufacturer. For example, a user's name or food preferences and the amount of memory on device are defined context information. Thus the defined value may change, but is often relatively static.
- *Aggregated* context information is obtained by joining, or aggregating, sensed or defined context. For example, family food preferences can be aggregated from family members' food preferences.
- *Deduced* context information can be obtained by using a context reasoning engine.

These are implemented with the additional *owl:classifiedAs* OWL property.

Also, quality of the context information is modeled by using four types of quality parameters:

- Accuracy: the range in terms of measurement
- Resolution: the smallest perceivable element
- Certainty: the probability of the state being certain
- Freshness: the production time and average lifetime of a measurement

Dependencies between different bits of context information are captured with the additional property *rdfs:dependsOn*. For example, in Figure 8.1 the feasibility of some scheduled activity is dependent on the person's location and weather, and the feasibility is modeled by the *feasible* property of the *ScheduledActivity* class.

```
<owl:ObjectProperty rdf:ID="feasible">
  <rdfs:domain rdf:resource="ScheduledActivity"/>
  <rdfs:dependsOn rdf:resource="locatedAt"/>
  <rdfs:dependsOn rdf:resource="weatherCondition"/>
</owl:ObjectProperty>
```

Figure 8.1: An Example of expressing a dependency with OWL

Context Reasoning

Reasoning in SOCAM is divided into two parts: ontology reasoning and user-defined reasoning. CONON uses only the axioms entailed by OWL Lite, so ontology reasoning can be done relatively efficiently. On the other hand, more flexible first-order-logic-based user-defined reasoning is available for reasoning about high-level contexts like “what is the user’s activity now”. User-defined reasoning resembles the reasoning of GAIA a lot, but as the rule base is divided, and e.g. reasoning about location may in many situations be feasible by using description logic, reasoning in whole can be much more efficient in general.

This distinction between two different reasoning methods is interesting, as it reminds of the distinction of the CoBrA Context Reasoning Engine introduced earlier and thus further reinforces the idea that context reasoning is best achieved by intervening more than one reasoning methods.

Interesting test results obtained by using standard home PCs and the whole CYC Upper Ontology merged with CONON are given. They show that with this setup reasoning for non-time-critical applications is feasible. On the other hand, real-time requirements for time-critical reasoning like phone call redirection are not really feasible without limiting the size of context datasets taken into account.

8.6 Discussion

A key advantage of using a formal model for context is that one can clearly specify the power and expressiveness of the model. In other words, it is clear what kinds of contexts can or cannot be expressed, what kinds of rules can or cannot be evaluated, and which queries are decidable and which are not [RC03]. On the other hand, understanding context and its meaning *in respect to some problem domain* better is needed in order to be able to assemble meaningful ontologies for different domains.

Context awareness can be, and indeed for many applications is, gradual in the sense that the amount of relevant information found and deduction that can be done based on it often determines the quality of adaptation. It is also clear that the definition of context by Dey et al is not far fetched at all. Thus the challenge for context modeling is that it should be possible to infer what information is relevant at a given context. This presents a big problem as the relevancy of the given information might depend on various things, and the relevancy can also change very rapidly. Thus a truly sophisticated context-aware system should try to anticipate the changes in context as well as possible in order to adapt to the new situation in a timely manner. Although these themes have been under investigation e.g. in situation awareness (SAW) systems [MKBL03], the current state of the art still has much room for improvement. The difference between situation awareness and context awareness is subtle. One view point is that “situation” is usually understood as more specific to some problem domain, the domain being usually characterized by time-critical tasks.

Although Gu et al state that “it would be easy to specify the context in one domain in which a specific range of context is of interest” [GWPZ04], it is far from clear how this range can be specified for different application domains in such a way that the reasoning is feasible and also produces worthwhile results. The “range of interest” can in fact be identified as the relevancy (‘dependency’ is sometimes used synonymously) of a given bit of context information, and may thus change rapidly. Further, relevancy itself is context dependent. For this reason it could be that relevancy is best modeled independently of formal reasoning, and only fairly static dependencies are modeled by the same representation.

It is clear that many times logical deduction will produce erroneous results; the world and human behavior are too complex to ever be captured perfectly by logics. If the applications do not have any indication of the quality, trustworthiness, and reliability of the context information, they will produce erroneous results too often to be “intelligent”, or even usable. Thus these dimensions of information should be modeled somehow. An open question is how this is done in the best, or even a “good enough”, way. We believe that for many applications the utility and cost of performing actions (or not performing them) based on reasoning should be incorporated to models somehow.

From the architectural point of view, perhaps the main result is that the presented formalisms seem to be in general relatively architecture-independent. This suggests that we can, at least to some extent, separate the concerns of representing context information and maintaining it between different system components and systems. The maintenance within one component is of course dependent on the representation of information, but the work of e.g. Tazari [Taz03] suggests that this might not be a major problem.

Concerning the mobile environment, a particular concern is the possibility to bound the amount of facts that are taken into account when inferring. There are basically three ways to do this. The first one is to create and use domain-specific ontologies always when they are available. These domain ontologies should pose knowledge of the domain in a “stricter package” than general ontologies can. The second way is to bound the amount of context information considered when reasoning, e.g. by using some estimation of relevancy. The third is to discard some facts based on their weak quality of information, but this last way does not help much in the case of a large ontology.

The research of ontologies, more specifically the evolution and management of dependent ontologies in a distributed environment, is crucial for large scale context-aware systems. As the vast majority of context representation formalisms expressive enough for reasoning have been formed during the last and this year, it is clear that there exist many research opportunities for forming ontologies and models suitable for specific problem domains. The maturation of the Semantic Web tools will likely enable reasonably good interoperability, although distributed ontology evolution still has a long road ahead of it before it can be done automatically.

Bibliography

[AAH+02] Heikki Ailisto, Petteri Alahuhta, Ville Haataja, Vesa Kyllönen, and Mikko Lindholm. Structuring context aware applications: Five-layer model and example case. Concepts and Models for Ubiquitous Computing Workshop, September 2002. (Position paper) http://www.comp.lancs.ac.uk/computing/users/dixa/conf/ubicomp2002-models/papers-list.html.

[ABG02] Joe Abley, Benjamin Black, and Vijay Gill. *Requirements for IPv6 Site-Multihoming Architectures*, May 2002. [Internet Draft] http://www.ietf.org/internet-drafts/draft-ietf-multi6-multihoming-requirements-03.txt.

[AS01a] Knarig Arabshian and Henning Schulzrinne. SIP-based emergency notification system. Technical report, Department of Computer Science, Columbia University, November 2001. http://www1.cs.columbia.edu/~knarig/EmergencyAlert.pdf.

[AS01b] Olivier Avaro and Philippe Salembier. MPEG-7 systems: Overview. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):760–764, June 2001.

[AS03] Knarig Arabshian and Henning Schulzrinne. A SIP-based medical event monitoring system. In *5th International Workshop on Enterprise Networking and Computing in Healthcare Industry*, June 2003. http://www1.cs.columbia.edu/~knarig/sipMed.pdf.

[ASC+00] Bob Aiken, John Strassner, Brian E. Carpenter, Ian Foster, Clifford Lynch, Joe Mambretti, Reagan Moore, and Benjamin Teitelbaum. *RFC 2768: Network Policy and Services: A Report of a Workshop on Middleware*. Internet Engineering Task Force, February 2000.

[Bar01] Dave Bartlett. CORBA junction: CORBA 3.0 notification service, May 2001. http://www-106.ibm.com/developerworks/webservices/library/co-cjct8/.

[BB02] Guruduth Banavar and Abraham Bernstein. Software infrastructure and design challenges for ubiquitous computing applications. *Communications of the ACM*, 45(12):92–96, December 2002.

[BBHS00] Peter Braam, Robert Baron, Jan Harkes, and Marc Schnieder. *The Coda HOWTO, version 1.01*, January 2000. http://www.coda.cs.cmu.edu/doc/html/coda-howto.html.

[BDNFT00] Giovanni Bricconi, Elisabetta Di Nitto, Alfonso Fuggetta, and Emma Tracanella. Analyzing the behavior of event dispatching systems through simulation. In *Proceedings of the 7th International Conference on High Performance Computing*, pages 131–140, December 2000.

[BDNT00] Giovanni Bricconi, Elisabetta Di Nitto, and Emma Tracanella. Issues in analyzing the behavior of event dispatching systems. In *Proceedings of the 10th International Workshop on Software Specification and Design*, page 95, November 2000.

[BKS$^+$99] Guruduth Banavar, Marc Kaplan, Kelly Shaw, Robert E. Strom, Daniel C. Sturman, and Wei Tao. Information flow based event distribution middleware. In Wei Sun, Sam Chanson, Doug Tygar, and Partha Dasgupta, editors, *ICDCS Workshop on Electronic Commerce and Web-based Applications*, pages 114–121, June 1999.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[BLR$^+$04] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. A layered naming architecture for the internet. In *ACM SIGCOMM 2004*, September 2004. http://nms.lcs.mit.edu/papers/layerednames-sigcomm04.pdf.

[BMH$^+$00] Jean Bacon, Ken Moody, Richard Hayton, et al. Generic support for distributed applications. *IEEE Computer*, 33(3):68–76, March 2000.

[BMT04] BEA, Microsoft, TIBCO. *Web Services Eventing (WS-Eventing)*, January 2004. http://xml.coverpages.org/WS-Eventing200401.pdf.

[BN99] Peter Braam and Philip Nelson. Removing bottlenecks in distributed filesystems: Coda and InterMezzo as examples. In *Proceedings of Linux Expo 1999*, May 1999. http://www.inter-mezzo.org/docs/bottlenecks.pdf.

[BP98] Sundar Balasubramaniam and Benjamin C. Pierce. What is a file synchronizer? In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 98–108, October 1998.

[BPSK97] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, December 1997.

[Bra98] Peter Braam. The Coda distributed file system. *Linux Journal*, 50, June 1998.

[Bra02] Peter Braam. InterMezzo: File synchronization with InterSync, version 0.9.3, March 2002. http://www.inter-mezzo.org/docs/intersync.pdf.

[BSSW03] Stefan Berger, Henning Schulzrinne, Stylianos Sidiroglou, and Xiaotao Wu. Ubiquitous computing using SIP. In *Proceedings of the 13th International workshop on Network and operating systems support for digital audio and video*, pages 82–89, June 2003. http://www.cs.columbia.edu/IRT/papers/Berg0306_Ubiquitous.pdf.

[Buc02] David Buchmann. SyncML (Synchronization Markup Language) and its Java implementation sync4j. Master's thesis, University of Fribourg, Fribourg, Switzerland, September 2002.

[Cam01] Gonzalo Camarillo. *SIP Demystified*. McGraw-Hill, New York, New York, August 2001.

[Cam03] Stefano Campadello. *Middleware Infrastructure for Distributed Mobile Applications*. PhD thesis, University of Helsinki, Department of Computer Science, Helsinki, Finland, April 2003. `http://ethesis.helsinki.fi/julkaisut/mat/tieto/vk/campadello/`.

[CCW03] Mauro Caporuscio, Antonio Carzaniga, and Alexander L. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. Technical Report CU-CS-944-03, Department of Computer Science, University of Colorado, January 2003.

[CDKR02] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication*, 20(8), October 2002.

[CDN01] Gianpaolo Cugola and Elisabetta Di Nitto. Using a publish/-subscribe middleware to support mobile computing. In *Middleware for Mobile Computing Workshop*, November 2001.

[CDNF01] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, September 2001.

[CDNP00] Gianpaolo Cugola, Elisabetta Di Nitto, and Gian Pietro Picco. Content-based dispatching in a mobile environment. In *Workshop su Sistemi Distribuiti: Algorithmi, Architectture e Linguaggi*, September 2000.

[CDW01] Antonio Carzaniga, Jing Deng, and Alexander L. Wolf. Fast forwarding for content-based networking. Technical Report CU-CS-922-01, Department of Computer Science, University of Colorado, November 2001.

[CFJ03] Harry Chen, Tim Finin, and Anupam Joshi. An intelligent broker for context-aware systems. In *Adjunct Proceedings of Ubicomp 2003*, pages 183–184, October 2003.

[CFJ04] Harry Chen, Tim Finin, and Anupam Joshi. A context broker for building smart meeting rooms. In *Proceedings of the Knowledge Representation and Ontology for Autonomous*

Systems Symposium, 2004 AAAI Spring Symposium, March 2004.

[Chi99] J. Noel Chiappa. *Endpoints and Endpoint Names: A Proposed Enhancement to the Internet Architecture*, 1999. http://users.exis.net/~jnc/tech/endpoints.txt.

[CHKT03] Scott Cantor, Jeff Hodges, John Kemp, and Peter Thompson. *Liberty ID-FF Architecture Overview, version 1.2*. Liberty Alliance, 2003. http://www.projectliberty.org/specs/liberty-idff-arch-overview-v1.2.pdf.

[Cis03] Cisco Systems. *Cisco IOS SIP Configuration Guide*, November 2003. http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123cgcr/vvfax_c/callc_c/sipc1_c/sipconf.pdf.

[CRW99] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Interfaces and algorithms for a wide-area event notification service. Technical Report CU-CS-888-99, Department of Computer Science, University of Colorado, October 1999. revised May 2000.

[CSZ03] Yuan Chen, Karsten Schwan, and Dong Zhou. Opportunistic channels: Mobility-aware event delivery. In *ACM/IFIP/USENIX International Middleware Conference 2003*, pages 182–201, June 2003. http://link.springer.de/link/service/series/0558/bibs/2672/26720182.htm.

[CW01] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, October 2001.

[CW03] Antonio Carzaniga and Alexander L. Wolf. Forwarding in a content-based network. In *Proceedings of 2003 conference on Applications, technologies, architectures, and protocol for computer communications*, pages 163–174, August 2003.

[DA99] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology, 1999. ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf.

[DAMV00] Mark Day, Sonu Aggarwal, Gordon Mohr, and Jesse Vincent. *RFC 2779: Instant Messaging / Presence Protocol Requirements*. Internet Engineering Task Force, February 2000. http://www.ietf.org/rfc/rfc2779.txt.

[Dey01] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, February 2001.

[DRS00] Mark Day, Jonathan Rosenberg, and Hiroyasu Sugano. *RFC 2778: A Model for Presence and Instant Messaging*. Internet Engineering Task Force, February 2000. http://www.ietf.org/rfc/rfc2778.txt.

[DSBE03] John S. Davis, Daby M. Sow, Marion Blount, and Maria R. Ebling. Context tailor: Towards a programming model for context-aware computing. In *1st International ACM Workshop on Middleware for Pervasive and Ad-Hoc Computing*, pages 68–75, June 2003.

[DSDE03] John S. Davis, Daby M. Sow, Angela B. Dalton, and Maria R. Ebling. Context-sensitive invocation using the context tailor infrastructure. In *Fifth Annual Conference on Ubiquitous Computing, October 2003*, October 2003.

[DZD$^+$03] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common API for structured peer-to-peer networks. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, volume 2735 of *Lecture Notes in Computer Science*, pages 33–44. Springer, February 2003. http://oceanstore.cs.berkeley.edu/publications/.

[E$^+$97] W. Keith Edwards et al. Designing and implementing asynchronous collaborative applications with Bayou. In *Proceedings of 10th annual ACM Symposium on User Interface Software and Technology*, pages 119–128, October 1997.

[EBDN03] Keith Edwards, Victoria Bellotti, Anind K. Dey, and Mark Newman. Stuck in the middle: The challenges of user-centered design and evaluation for middleware. In *Conference on Human Factors in Computing Systems*, pages 297–304, apr 2003.

[EBS01] Greg Eisenhauer, Fabián Bustamante, and Karsten Schwan. A middleware toolkit for client-initiated service specialization. *ACM SIGOPS Operating Systems Review*, 35(2):7–20, April 2001.

[Egg04] Lars Eggert. *Host Identity Protocol (HIP) Rendezvous Mechanisms*. Internet Engineering Task Force, February 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-eggert-hip-rendezvous-00.txt.

[EL04a] Lars Eggert and J. Laganier. *Host Identity Protocol (HIP) Rendezvous Extensions*. Internet Engineering Task Force, July 2004. [Internet Draft], Work in progress, http://www.ietf.org/internet-drafts/draft-eggert-hip-rvs-00.txt.

[EL04b] Lars Eggert and M. Liebsch. *Design Aspects of Host Identity Protocol (HIP) Rendezvous Mechanism*. Internet Engineering Task Force, July 2004. [Internet Draft], Work in progress, http://www.ietf.org/internet-drafts/draft-eggert-hip-rendezvous-01.txt.

[Ere01] Justin R. Erenkrantz. Handling hierarchical events in an internet-scale event service, March 2001. http://www.ucf.ics.uci.edu/~jerenk/siena-xml/SienaPaper.html.

[Fan02] Nicola Fankhauser. A real world application of SyncML based on open source components. Seminar work, University of Fribourg, Fribourg, Switzerland, December 2002.

[FGKZ03] Ludger Fiege, Felix C. Gartner, Oliver Kasten, and Andreas Zeidler. Supporting mobility in content-based publish/subscribe middleware, June 2003. http://citeseer.nj.nec.com/kasten03supporting.html.

[FGM+99] Roy Fielding, James Gettys, Jeffrey Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul Leach, and Tim Berners-Lee. *RFC 2616: Hypertext Transfer Protocol — HTTP/1.1*. Internet Engineering Task Force, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[FIP01] Foundation for Intelligent Physical Agents, Geneva, Switzerland. *FIPA Ontology Service Specification*, August 2001. http://www.fipa.org/.

[FIP02] Foundation for Intelligent Physical Agents, Geneva, Switzerland. *FIPA ACL Message Structure Specification*, December 2002. http://www.fipa.org/.

[FY01] Gerhard Fischer and Yunwen Ye. Exploiting context to make delivered information relevant to tasks and users. In *Workshop on User Modelling for Context-Aware Applications, 8th International Conference on User Modeling*, July 2001.

[GA02] Jean-loup Gailly and Mark Adler. *zlib 1.1.4 Manual*, March 2002. http://www.gzip.org/zlib/manual.html.

[GCSO01] Pradeep Gore, Ron Cytron, Douglas Schmidt, and Carlos O'Ryan. Designing and optimizing a scalable CORBA notification service. *ACM SIGPLAN Notices*, 36(8):196–204, August 2001.

[GdMCA02] Hrishikesh Gossain, Carlos de Morais Cordeiro, and Dharma P. Agrawal. Multicast: Wired to wireless. *IEEE Communications Magazine*, 40(6):116–123, June 2002.

[GM02] Miguel Garcia-Martin. *3rd-Generation Partnership Project (3GPP) Release 5 requirements on the Session Initiation Protocol (SIP)*. Internet Engineering Task Force, October 2002. [Internet Draft] http://www.iptel.org/info/players/ietf/3gpp/draft-ietf-sipping-3gpp-r5-requirements-00.txt.

[Gru93] Tom Gruber. A translation approach to portable ontologies. *Journal of Knowledge Acquisition*, 5(2):199–220, June 1993.

[GS00] Marc Girardot and Neel Sundaresan. Millau: an encoding format for efficient representation and exchange of XML over the Web. In *Ninth International World Wide Web Conference*, May 2000. http://www9.org/w9cdrom/154/154.html.

[GWPZ04] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. An ontology-based context model in intelligent environments. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, January 2004.

[HBS02] Albert Held, Sven Buchholz, and Alexander Schill. Modeling of context information for pervasive computing applications. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics*, July 2002.

[HC98] Dan Harkins and Dave Carrel. *RFC 2409: The Internet Key Exchange (IKE)*. Internet Engineering Task Force, November 1998. http://www.ietf.org/rfc/rfc2409.txt.

[HD02] Klaus Marius Hansen and Christian Heide Damm. Instant collaboration — using context-aware instant messaging for session management in distributed collaboration tools. In *Proceedings of the second Nordic conference on Human-computer interaction*, pages 279–282, October 2002.

[Hei01] Dennis Heimbigner. Adapting publish/subscribe middleware to achieve Gnutella-like functionality. In *Proceedings of the 2001 ACM Symposium on Applied Computing*, pages 176–181, March 2001.

[Hen03] Thomas R. Henderson. Host mobility for IP networks: A comparison. *IEEE Network Magazine*, 17(6):18–26, November 2003.

[HKZ02] Abdelsalam Helal, Abhinav Khushraj, and Jinsuo Zhang. Incremental hoarding and reintegration in mobile environments. In *Proceedings of the 2002 Symposium on Applications and the Internet*, pages 8–11, February 2002.

[HM02] Jeff Hodges and Bob Morgan. *RFC 3377: Lightweight Directory Access Protocol (v3): Technical Specification*. Internet Engineering Task Force, September 2002. http://www.ietf.org/rfc/rfc3377.txt.

[HMN$^+$00] Mads Haahr, Rene Meier, Paddy Nixon, Vinny Cahill, and Eric Jul. Filtering and scalability in the ECO distributed event model. In *PDSE*, pages 83–95, 2000.

[Hob02] Jerry R. Hobbs. A DAML ontology of time, November 2002. http://www.cs.rochester.edu/~ferguson/daml/daml-time-20020830.txt.

[HP94] John S. Heidemann and Gerald J. Popek. File-system development with stackable layers. *ACM Transactions on Computer Systems*, 12(1):58–89, February 1994.

[HPR89] Susan Horwitz, Jan Prins, and Thomas Reps. Integrating noninterfering versions of programs. *ACM Transactions on*

Programming Languages and Systems, 11(3):345–387, July 1989.

[HS96] Scott E. Hudson and Ian Smith. Techniques for addressing fundamental privacy and disruption tradeoffs in awareness support systems. In *Conference of Computer Supported Co-operative Work '96, ACM*, pages 248–257, November 1996.

[HSSR99] Mark Handley, Henning Schulzrinne, Eve Schooler, and Jonathan Rosenberg. *RFC 2543: SIP: Session Initiation Protocol*. Internet Engineering Task Force, March 1999. http://www.ietf.org/rfc/rfc2543.txt.

[IBM02a] IBM. *Gryphon: Publish/subscribe over public networks.*, December 2002. (White paper) http://www.research.ibm.com/gryphon/Gryphon/Gryphon-Overview.pdf.

[IBM02b] IBM. *MQSeries Everyplace for Multiplatforms Version 1, Release 2*, 2002. (White paper) http://www-3.ibm.com/software/ts/mqseries/everyplace/v12/whitepaper.html.

[ION02] IONA. *CORBA Notification Server Guide, version 5.1*, April 2002. http://www.iona.com/support/docs/e2a/asp/5.0/enterprise.xml.

[IRD03] Infrared Data Association (IrDA). *Object Exchange Protocol OBEX*, January 2003. http://www.irda.org/standards/specifications.asp.

[ISO86] International Organization for Standardization, Geneva, Switzerland. *ISO 8879:1986. Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, 1986.

[IWR02] Ellen Isaacs, Alan Walendowski, and Dipti Ranganathan. Hubbub: A sound-enhanced mobile instant messenger that supports awareness and opportunistic interactions. In *Proceedings of the Conference on Computer-Human Interaction*, pages 179–186, April 2002.

[IWW$^+$02] Ellen Isaacs, Alan Walendowski, Steve Whittaker, Diane J. Schiano, and Candace Kamm. The character, functions, and styles of instant messaging in the workplace. In *Proceedings* *of the 2002 ACM conference on Computer supported cooperative work*, pages 11–20. ACM Press, November 2002.

[JBA01] Ravi Jain, John-Luc Bakker, and Farooq Anjum. Java Call Control (JCC) and Session Initiation Protocol. *IEICE Transactions on communications*, E84-B(12), December 2001.

[Jel02] Rick Jelliffe. *The Schematron Assertion Language 1.5*. Academia Sinica Computing Centre, October 2002. http://xml.ascc.net/resource/schematron/Schematron2000.html.

[JHE99] Jin Jing, Abdelsalam Helal, and Ahmed Elmagarmid. Client-server computing in mobile environments. *ACM Computing Surveys*, 31(2):117–157, June 1999.

[JN02] Christophe Jelger and Thomas Noel. Multicast for mobile hosts in IP networks: Progress and challenges. *IEEE Wireless Communications*, 9(5), October 2002.

[Jus01] Jari Juslin. Navigointi WAP-sovelluksissa. Master's thesis, University of Helsinki, Department of Computer Science, Helsinki, Finland, August 2001.

[K$^+$00] John Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[KA98a] Stephen Kent and Randall Atkinson. *RFC 2401: Security Architecture for the Internet Protocol*. Internet Engineering Task Force, November 1998. http://www.ietf.org/rfc/rfc2401.txt.

[KA98b] Stephen Kent and Randall Atkinson. *RFC 2406: IP Encapsulating Security Payload (ESP)*. Internet Engineering Task Force, November 1998. http://www.ietf.org/rfc/rfc2406.txt.

[Kar03] Gerald Karner. A novel call control system for broadband satellite systems. In *IP Networking over Satellite Workshop 2003*, May 2003. http://telecom.esa.int/telecom/media/document/karner.pdf.

[KFJ03] Lalana Kagal, Tim Finin, and Anupam Josh. A policy language for a pervasive computing environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pages 63–76, June 2003.

[Kis01] Roman Kiss. Using the COM+ event system in .Net applications, September 2001. `http://www.codeproject.com/csharp/solutionlcenotification.asp`.

[Kiv04] Tero Kivinen. *MOBIKE protocol*, February 2004. [Internet Draft] `http://www.ietf.org/internet-drafts/draft-kivinen-mobike-protocol-00.txt`.

[KMP$^+$00] Malleswar Kalla, Ken Morneault, Vern Paxson, Ian Rytina, Hanns Jürgen Schwarzbauer, Chip Sharp, Randall Stewart, Tom Taylor, Qiaobing Xie, and Lixia Zhang. *RFC 2960: Stream Control Transmission Protocol*. Internet Engineering Task Force, October 2000. `http://www.ietf.org/rfc/rfc2960.txt`.

[Kom02] Miika Komu. Host identity payload in home networks. Seminar paper, Helsinki University of Technology, Espoo, Finland, April 2002.

[Kom04] Miika Komu. Application programming interfaces for the host identity protocol. Master's thesis, Helsinki University of Technology, September 2004. `http://hipl.hiit.fi/hipl/hip-native-api-final.pdf`.

[Kra03] Hugo Krawczyk. *SIGMA: 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols*, June 2003. `http://www.ee.technion.ac.il/~hugo/sigma.ps`.

[KTR03] Jaakko Kangasharju, Sasu Tarkoma, and Kimmo Raatikainen. Comparing SOAP performance for various encodings, protocols, and connections. In Marco Conti, Silvia Giordano, Enrico Gregori, and Stephan Olariu, editors, *Personal Wireless Communications*, pages 397–406, September 2003.

[Kub03] John Kubiatowicz. Extracting guarantees from chaos. *Communications of the ACM*, 46(2):33–38, February 2003.

[LGS97] Alison Lee, Andreas Girgensohn, and Kevin Schlueter. Nynex portholes: Initial user reactions and redesign implications. In *GROUP'97, International Conference on Supporting Group Work*, pages 385–394. ACM Press, 1997.

[LH03] Mikko Laukkanen and Heikki Helin. Web services in wireless networks — what happened to the performance? In Liang-Jie Zhang, editor, *Proceedings of the International Conference on Web Services*, pages 278–284, June 2003.

[LHKR96] Mika Liljeberg, Heikki Helin, Markku Kojo, and Kimmo Raatikainen. Mowgli WWW software: improved usability of WWW in mobile WAN environments. In *Proceedings of IEEE Global Internet 1996*, pages 33–37, November 1996.

[Li00] Sing Li. *Professional Jini*. Wrox Press, Birmingham, United Kingdom, 2000.

[Lib03] Liberty Alliance Project. *Introduction to the Liberty Alliance Identity Architecture, version 1.0*, March 2003. http://www.projectliberty.org/resources/whitepapers/LAP%20Identity%20Architecture%20Whitepaper%20Final.pdf.

[Lin03] Tancred Lindholm. XML three-way merge as a reconciliation engine for mobile data. In *Third ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 93–97, September 2003.

[LLS02] Yui-Wah Lee, Kwong-Sak Leung, and Mahadev Satyanarayanan. Operation shipping for mobile file systems. *IEEE Transactions on Computers*, 51(12):1410–1422, December 2002.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[LV95] David C. Luckham and James Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, 21(9):717–734, September 1995. http://citeseer.nj.nec.com/luckham95eventbased.html.

[MC03] René Meier and Vinny Cahill. Exploiting proximity in event-based middleware for collaborative mobile applications. In *4th International Conference on Distributed Applications and Interoperable Systems*, pages 285–296, November 2003. http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2893&spage=285.

[McG02] Deborah L. McGuinness. Ontologies come of age. In Dieter Fensel, James Hendler, Henry Lieberman, and Wolfgang Wahlster, editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, November 2002.

[Mei00] René Meier. State of the art review of distributed event models. Technical Report TCD-CS-2000-15, Department of Computer Science, Trinity College, Dublin, Ireland, March 2000. http://citeseer.nj.nec.com/437791.html.

[MES95] Lily Mummert, Maria Ebling, and Mahadev Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles*, pages 143–155, December 1995.

[Mic99a] Microsoft. *Introduction to IntelliMirror Management Technologies*, 1999. White paper.

[Mic99b] Microsoft. *Message Queuing on Windows CE*, June 1999. Windows CE Developers Conference, http://www.microsoft.com/msmq/downloads/devcon99.ppt.

[Mic02] Microsoft. *Message Queuing in Windows XP: New Features*, 2002. (White paper) http://www.microsoft.com/msmq/MSMQ3.0_whitepaper_draft.doc.

[MKBL03] Christopher J. Matheus, Mieczyslaw M. Kokar, Kenneth Baclawski, and Jerzy Letkowski. Constructing RuleML-based domain theories on top of OWL ontologies. In *Rules and Rule Markup Languages for the Semantic Web*, volume 2876 of *Lecture Notes in Computer Science*, pages 81–94. Springer-Verlag, Heidelberg, Germany, November 2003.

[MN03] Robert Moskowitz and Pekka Nikander. *Host Identity Payload Architecture*. Internet Engineering Task Force, September 2003. [Internet Draft] http://www.ietf.org/internet-drafts/draft-moskowitz-hip-arch-05.txt.

[MNJH04a] Robert Moskowitz, Pekka Nikander, Petri Jokela, and Thomas Henderson. *Host Identity Protocol*. Internet Engineering Task Force, February 2004. [Internet Draft] http://hip4inter.net/documentation/drafts/draft-moskowitz-hip-09.txt.

[MNJH04b] Robert Moskowitz, Pekka Nikander, Petri Jokela, and Thomas Henderson. *Host Identity Protocol*. Internet Engineering Task Force, June 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-ietf-hip-base-00.txt.

[Mob01] Mobiliti. *Overview of Intelligent Delta Selection Process (iDESP)*, August 2001. http://www.mobiliti.com/PDF/iDESPOverview30.pdf.

[Moc87] Paul Mockapetris. *RFC 1034: Domain Names — Concepts and Facilities*. Internet Engineering Task Force, November 1987. http://www.ietf.org/rfc/rfc1034.txt.

[Mos01] Robert Moskowitz. *Host Identity Payload Implementation*. Internet Engineering Task Force, February 2001. [Internet Draft] http://homebase.htt-consult.com/~hip/draft-moskowitz-hip-impl-01.txt.

[MS91] Henry Mashburn and Mahadev Satyanarayanan. *RVM: Recoverable Virtual Memory User Manual*, April 1991.

[NA04a] Pekka Nikander and Jari Arkko. *End-Host Mobility and Multi-Homing with Host Identity Protocol*. Internet Engineering Task Force, January 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-nikander-hip-mm-02.txt.

[NA04b] Pekka Nikander and Jari Arkko. *Host Identity Indirection Infrastructure (Hi3)*. Internet Engineering Task Force, June 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-nikander-hiprg-hi3-00.txt.

[NH04] Pekka Nikander and Thomas Henderson. *Considerations on HIP based IPv6 multi-homing*. Internet Engineering Task Force, July 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-nikander-multi6-hip-01.txt.

[Nik02] Pekka Nikander. A case for host identity payload: An architecture for multihomed mobile hosts, February 2002. unpublished manuscript.

[NIS94] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*, May 1994. http://www.itl.nist.gov/fipspubs/fip186.htm.

[NL04] P. Nikander and J. Laganier. *Host Identity Protocol (HIP) Domain Name System (DNS) Extensions*. IETF, May 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-nikander-hip-dns-00.txt.

[Now89] Bill Nowicki. *RFC 1094: NFS Network File System Protocol Specification*. Internet Engineering Task Force, March 1989. http://www.ietf.org/rfc/rfc1094.txt.

[NYJW04] Pekka Nikander, Jukka Ylitalo, Petri Jokela, and Jorma Wall. *Integrating Security, Mobility, and Multihoming in a HIP Way*, February 2004. http://www.tml.hut.fi/~pnr/publications/NDSS03-Nikander-et-al.pdf.

[OAS01] OASIS. *RELAX NG Specification*, December 2001. http://www.relaxng.org/spec-20011203.html.

[OAS02a] OASIS. *UDDI Version 3.0*, July 2002. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.

[OAS02b] OASIS. *Message Service Specification, Version 2.0*, April 2002. http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf.

[OAS04] OASIS. *Web Services Security: SOAP Message Security 1.0*, March 2004. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.

[O'D03] Phelim O'Doherty. SIP and the Java platforms, June 2003. http://java.sun.com/products/jain/SIP-and-Java.html.

[OMA03] Open Mobile Alliance. *WV-041 Features and Functions, version 1.2*, February 2003.

[OMG01a] Object Management Group. *CORBA Event Service Specification v.1.1*, March 2001.

[OMG01b] Object Management Group. *CORBA Notification Service Specification v.1.0*, March 2001.

[OMG01c] Object Management Group. *Management of Event Domains Specification*, June 2001. http://www.omg.org/cgi-bin/doc?formal/2001-06-03.

[OMG02] Object Management Group. *Joint Initial Submission regarding the JMS Notification Service RFP*, January 2002. http://www.omg.org/cgi-bin/doc?telecom/02-01-02.

[OR93] Jarkko Oikarinen and Darren Reed. *RFC 1459: Internet Relay Chat Protocol*. Internet Engineering Task Force, May 1993. http://www.ietf.org/rfc/rfc1459.txt.

[OR02] Eamon O'Tuathail and Marshall T. Rose. *RFC 3288: Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP)*. Internet Engineering Task Force, June 2002. http://www.ietf.org/rfc/rfc3288.txt.

[P$^+$83] Douglas Stott Parker, Jr. et al. Detection of mutual inconsistency in distributed systems. *IEEE Transactions on Software Engineering*, 9(3):240–247, May 1983.

[Pab02] Chandandeep Pabla. SyncML intensive — a beginner's look at the SyncML protocol and procedures, April 2002. http://www-106.ibm.com/developerworks/wireless/library/i-syncml2/.

[Per96] Charles Perkins. *RFC 2002: IP Mobility Support*. Internet Engineering Task Force, October 1996. http://www.ietf.org/rfc/rfc2002.txt.

[Pla99] David Platt. The COM+ event service eases the pain of publishing and subscribing to data. *Microsoft Systems Journal*, September 1999. http://www.microsoft.com/msj/0999/comevent/comevent.aspx.

[Pos80] Jon Postel. *RFC 768: User Datagram Protocol*. Internet Engineering Task Force, August 1980. http://www.ietf.org/rfc/rfc768.txt.

[Pos81] Jon Postel. *RFC 793: Transmission Control Protocol*. Internet Engineering Task Force, September 1981. http://www.ietf.org/rfc/rfc793.txt.

[Pri01] Prism Technologies. *Open Fusion Notification Service*, May 2001. (White paper) `http://www.prismtechnologies.com/English/Products/CORBA/CORBA_services/notification/whitepaper/01_Notification_may_01.html`.

[PRR97] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, June 1997.

[PSB03] Peter R. Pietzuch, Brian Shand, and Jean Bacon. A framework for event composition in distributed systems. In *Proceedings of the 4th International Conference on Middleware*, pages 62–82, June 2003. `http://citeseer.nj.nec.com/article/pietzuch03framework.html`.

[PST$^+$97] Karin Petersen, Mike Spreitzer, Douglas Terry, Marvin Theimer, and Alan J. Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the sixteenth ACM Symposium on Operating Systems Principles*, pages 288–301, September 1997.

[PW03] Birgit Pfitzmann and Michael Waidner. Analysis of liberty single-sign-on with enabled clients. *IEEE Internet Computing Magazine*, 7(6):38–44, November 2003.

[R$^+$01a] Bill Ray et al. *Professional Java Mobile Programming*. Wrox Press, Birmingham, United Kingdom, July 2001.

[R$^+$01b] Sean Rhea et al. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5):40–49, September 2001.

[RC03] Anand Ranganathan and Roy H. Campbell. An infrastructure for context-awareness based on first order logic. *Personal Ubiquitous Computing*, 7(6):353–364, December 2003.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, October 2001.

[REG+03] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: the OceanStore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, March 2003.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. *Computer Communication Review*, 31(4):161–172, October 2001.

[RGRK03] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowic. Handling churn in a DHT. Technical Report UCB//CSD-03-1299, The University of California, Berkeley, December 2003. http://bamboo-dht.org/pubs.html.

[RHC+02] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, October 2002.

[Rii03a] Pekka Riikonen. *Secure Internet Live Conferencing (SILC) Protocol Specification*, July 2003. [Internet Draft] http://www.ietf.org/internet-drafts/draft-riikonen-silc-spec-07.txt.

[Rii03b] Pekka Riikonen. *SILC Message Flag Payloads*, December 2003. [Internet Draft] http://www.ietf.org/internet-drafts/draft-riikonen-silc-flags-payloads-04.txt.

[Rii03c] Pekka Riikonen. *User Online Presence and Information Attributes*, July 2003. [Internet Draft] http://www.ietf.org/internet-drafts/draft-riikonen-presence-attrs-02.txt.

[Rii04a] Pekka Riikonen. *Secure Internet Live Conferencing (SILC), Protocol Specification*, February 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-riikonen-silc-spec-08.txt.

[Rii04b] Pekka Riikonen. *SILC Commands*, February 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-riikonen-silc-commands-06.txt.

[Rii04c] Pekka Riikonen. *SILC Key Exchange and Authentication Protocols*, February 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-riikonen-silc-ke-auth-08.txt.

[Rii04d] Pekka Riikonen. *SILC Packet Protocol*, February 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-riikonen-silc-pp-08.txt.

[RMCM03] Anand Ranganathan, Robert E. McGrath, Roy H. Campbell, and M. Dennis Mickunas. Ontologies in a pervasive computing environment. In *Workshop on Ontologies and Distributed Systems (part of the 18'th International Joint Conference on Artificial Intelligence)*, August 2003.

[Roa02] Adam Roach. *RFC 3265: Session Initiation Protocol (SIP)-specific Event Notification*. Internet Engineering Task Force, June 2002. http://www.ietf.org/rfc/rfc3265.txt.

[Ros01a] Marshall T. Rose. *RFC 3080: The Blocks Extensible Exchange Protocol Core*. Internet Engineering Task Force, March 2001. http://www.ietf.org/rfc/rfc3080.txt.

[Ros01b] David Rosenblum. A tour of Siena, an interoperability infrastructure for internet-scale distributed architectures. In *Ground System Architectures Workshop*, February 2001. http://sunset.usc.edu/GSAW/gsaw2001/SESSION3/Siena.pdf.

[Ros04] Jonathan Rosenberg. *A Presence Event Package for the Session Initiation Protocol (SIP)*. Internet Engineering Task Force, August 2004. http://www.ietf.org/rfc/rfc3856.txt.

[RSC+02] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. *RFC 3261: SIP: Session Initiation Protocol*. Internet Engineering Task Force, June 2002. http://www.ietf.org/rfc/rfc3261.txt.

[S+90] Mahadev Satyanaraynan et al. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[S+97] Mike J. Spreitzer et al. Dealing with server corruption in weakly consistent, replicated data systems. In *Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 234–240, September 1997.

[S+01] Tony Speakman et al. *RFC 3208: PGM Reliable Transport Protocol Specification*. Internet Engineering Task Force, December 2001. http://www.ietf.org/rfc/rfc3208.txt.

[SA04a] Peter Saint-Andre. *End-to-End Signing and Object Encryption in the Extensible Messaging and Presence Protocol (XMPP)*. Internet Engineering Task Force, July 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-ietf-xmpp-e2e-09.txt.

[SA04b] Peter Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. Internet Engineering Task Force, May 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-ietf-xmpp-core-24.txt.

[SA04c] Peter Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging*. Internet Engineering Task Force, April 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-ietf-xmpp-im-22.txt.

[SA04d] Peter Saint-Andre. *Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)*. Internet Engineering Task Force, May 2004. [Internet Draft] http://www.ietf.org/internet-drafts/draft-ietf-xmpp-cpim-05.txt.

[SAS01] Peter Sutton, Rhys Arkins, and Bill Segall. Supporting disconnectedness — transparent information delivery for mobile and invisible computing. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 277, May 2001.

[Sat96] Mahadev Satyanarayanan. Mobile information access. *IEEE Personal Communications*, 3(1):26–33, February 1996.

[Sau02] Christopher Saunders. A lack of SIMPLE pleasures, November 2002. http://www.instantmessagingplanet.com/enterprise/article.php/10816_1498911.

[SAZ$^+$02] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *ACM SIGCOMM 2002*, aug 2002. http://i3.cs.berkeley.edu/publications/papers/i3-sigcomm.pdf.

[SB00] Jürgen Schirmer and Holger Bach. Context management in an agent-based approach for service assistance in the domain of consumer electronics. In *Proceedings of the 2000 Conference on Intelligent Interactive Assistance and Mobile Multimedia Computing*, November 2000.

[SDA99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, May 1999.

[SGGB02] Aleksander Slominski, Madhusudhan Govindaraju, Dennis Gannon, and Randall Bramley. An extensible and interoperable event system architecture using SOAP. Technical Report TR549, Department of Computer Science, Indiana University, February 2002. http://www.extreme.indiana.edu/xgws/papers/events_paper/.

[Sie99] Jon Siegel. An overview of CORBA 3. In *Proceedings of the Second International Working Conference on Distributed Applications and Interoperable Systems*, July 1999.

[Sin01] Henry Sinnreich. *Internet Communications Using SIP: Delivering VoIP and Multimedia Services with Session Initiation Protocol*. Wiley, Hoboken, New Jersey, November 2001.

[SK92] Mahadev Satyanarayanan and James Kistler. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[SLPF03] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Cool: A context ontology language to enable contextual interoperability. In Jean-Bernard Stefani, Isabelle Dameure, and Daniel Hagimont, editors, *Proceedings of the 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, volume 2893 of *Lecture Notes in Computer Science*, pages 236–247. Springer-Verlag, November 2003.

[SMK+01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Computer Communication Review*, 31(4):149–160, October 2001.

[SPGK+03] Paul Sandoz, Santiago Pericas-Geertsen, Kohuske Kawaguchi, Marc Hadley, and Eduardo Pelegri-Llopart. Fast Web services, August 2003. http://developer.java.sun.com/developer/technicalArticles/WebServices/fastWS/index.html.

[SQ04] M. Stiemerling and J. Quittek. *Problem Statement: HIP operation over Network Address Translators*. Internet Engineering Task Force, February 2004. [Expired Internet Draft].

[Sri01] Paddy Srinivas. Introduction to COM+ events, March 2001. http://www.idevresource.com/com/library/articles/com+eventsintro.asp.

[SRL96] Kevin Savetz, Neil Randall, and Yves Lepage. *MBONE: Multicasting Tomorrow's Internet*, April 1996. http://www.savetz.com/mbone/.

[SS02] Yasushi Saito and Marc Shapiro. Replication: Optimistic approaches. Technical Report HPL-2002-33, Hewlett Packard Laboratories, February 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-33.pdf.

[Sun01] Sun Microsystems, Santa Clara, California, USA. *Java Message Service Specification*, June 2001.

[Sun04] Sun Microsystems, Santa Clara, California, USA. *JSR 172: J2ME Web Services Specification*, March 2004. http://jcp.org/aboutJava/communityprocess/final/jsr172/index.html.

[SW00] Henning Schulzrinne and Elin Wedlund. Application-layer mobility using SIP. *ACM SIGMobile*, 4(3):47–57, July 2000. http://doi.acm.org/10.1145/372346.372369.

[Syn01] SyncML Initiative. *The Business Case for Device Management*, November 2001. (White paper) http://www.syncml.org/syncml_devman_business_cases_whppr.pdf.

[Syn02a] SyncML Initiative. *SyncML Representation Protocol, version 1.1*, February 2002. http://www.syncml.org/docs/syncml_represent_v11_20020215.pdf.

[Syn02b] SyncML Initiative. *SyncML Sync Protocol, version 1.1*, February 2002. http://www.syncml.org/docs/syncml_sync_protocol_v11_20020215.pdf.

[Taz03] Mohammad-Reza Tazari. A context-oriented RDF database. In *Proceedings of the first International Workshop on Semantic Web and Databases*, pages 63–78, September 2003.

[TCH00] Telecordia Technologies Inc, Columbia University, Hughes Software Systems. *SIP Extensions for Communicating with Networked Appliances*, November 2000. http://www.hssworld.com/hss_mindsystem/ietf/sip_extn.htm.

[TD$^+$94] Douglas B. Terry, Alan J. Demers, , Karin Petersen, Mike Spreitzer, Marvin Theimer, and Brent W. Welch. Session guarantees for weakly-consistent replicated data. In *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems*, pages 140–149, September 1994.

[TGF03] Mohammad-Reza Tazari, Matthias Grimm, and Matthias Finke. Modeling user context. In *The 10th International Conference on Human-Computer Interaction (HCII)*, June 2003.

[Tri99] Andrew Tridgell. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University, Canberra, Australia, February 1999.

[TRTN03] Graham Thomson, Matthew Richmond, Sotirios Terzis, and Paddy Nixon. An approach to dynamic context discovery and composition. In *Proceedings of Ubisys: System Support for Ubiquitous Computing Workshop*, October 2003.

[vH02] Bill von Hagen. *Using the InterMezzo Distributed Filesystem — Getting Connected in a Disconnected World*, August 2002. http://www.linuxplanet.com/linuxplanet/reports/4368/1.

[W3C99a] World Wide Web Consortium. *HTML 4.01 Specification*, December 1999. [Recommendation] http://www.w3.org/TR/html401/.

[W3C99b] World Wide Web Consortium. *WAP Binary XML Content Format*, June 1999. [Note] http://www.w3.org/TR/wbxml/.

[W3C99c] World Wide Web Consortium. *Resource Description Framework (RDF) Model and Syntax Specification*, February 1999. [Recommendation] http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[W3C99d] World Wide Web Consortium. *XML Path Language (XPath) 1.0*, November 1999. [Recommendation] http://www.w3.org/TR/xpath.

[W3C00a] World Wide Web Consortium. *Document Object Model (DOM) Level 2 Events Specification, Version 1.0*, November 2000. [Recommendation] http://www.w3.org/TR/DOM-Level-2-Events/.

[W3C00b] World Wide Web Consortium. *Simple Object Access Protocol (SOAP) 1.1*, May 2000. [Note] http://www.w3.org/TR/SOAP/.

[W3C01a] World Wide Web Consortium. *XML Schema Part 1: Structures*, May 2001. [Recommendation] http://www.w3.org/TR/xmlschema-1/.

[W3C01b] World Wide Web Consortium. *XML Schema Part 2: Datatypes*, May 2001. [Recommendation] http://www.w3.org/TR/xmlschema-2/.

[W3C01c] World Wide Web Consortium. *Web Services Description Language (WSDL) 1.1*, March 2001. [Note] http://www.w3.org/TR/wsdl.

[W3C02a] World Wide Web Consortium. *SOAP Version 1.2 Email Binding*, June 2002. [Note] http://www.w3.org/TR/2002/NOTE-soap12-email-20020626.

[W3C02b] World Wide Web Consortium. *SOAP Version 1.2 Usage Scenarios*, June 2002. [Working Draft] http://www.w3.org/TR/2002/WD-xmlp-scenarios-20020626/.

[W3C02c] World Wide Web Consortium. *XML Encryption Syntax and Processing*, December 2002. http://www.w3.org/TR/xmlenc-core/.

[W3C02d] World Wide Web Consortium. *XML Signature Syntax and Processing*, February 2002. http://www.w3.org/TR/xmldsig-core/.

[W3C03a] World Wide Web Consortium. *OWL Web Ontology Language Guide*, December 2003. [Proposed Recommendation] http://www.w3.org/TR/owl-guide/.

[W3C03b] World Wide Web Consortium. *OWL Web Ontology Language Overview*, December 2003. [Proposed Recommendation] http://www.w3c.org/TR/owl-features/.

[W3C03c] World Wide Web Consortium. *OWL Web Ontology Language Semantics and Abstract Syntax*, December 2003. [Proposed Recommendation] http://www.w3c.org/TR/owl-semantics/.

[W3C03d] World Wide Web Consortium. *SOAP Version 1.2 Part 1: Messaging Framework*, June 2003. [Recommendation] http://www.w3.org/TR/soap12-part1/.

[W3C03e] World Wide Web Consortium. *SOAP Version 1.2 Part 2: Adjuncts*, June 2003. [Recommendation] http://www.w3.org/TR/soap12-part2/.

[W3C03f] World Wide Web Consortium. *SOAP Version 1.2 Specification Assertions and Test Collection*, June 2003. [Recommendation] http://www.w3.org/TR/2003/REC-soap12-testcollection-20030624/.

[W3C03g] World Wide Web Consortium. *XML Events — An Events Syntax for XML*, February 2003. [Candidate Recommendation] http://www.w3.org/TR/2003/CR-xml-events-20030207.

[W3C04a] World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, August 2004. [Last Call Working Draft] http://www.w3.org/TR/2004/WD-wsdl20-20040803.

[W3C04b] World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0 Part 2: Predefined Extensions*, August 2004. [Last Call Working Draft] http://www.w3.org/TR/2004/WD-wsdl20-extensions-20040803.

[W3C04c] World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0 Part 3: Bindings*, August 2004. [Last Call Working Draft] http://www.w3.org/TR/2004/WD-wsdl20-bindings-20040803.

[W3C04d] World Wide Web Consortium. *XML Binary Characterization Use Cases*, July 2004. [Working Draft] http://www.w3.org/TR/2004/WD-xbc-use-cases-20040728/.

[W3C04e] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0*, 3rd edition, February 2004. [Recommendation] http://www.w3.org/TR/2004/REC-xml-20040204/.

[W3C04f] World Wide Web Consortium. *Extensible Markup Language (XML) 1.1*, February 2004. [Recommendation] http://www.w3.org/TR/2004/REC-xml11-20040204/.

[W3C04g] World Wide Web Consortium. *SOAP Message Transmission Optimization Mechanism*, August 2004. [Candidate Recommendation] http://www.w3.org/TR/2004/CR-soap12-mtom-20040826/.

[W3C04h] World Wide Web Consortium. *Web Services Architecture*, February 2004. [Note] http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[W3C04i] World Wide Web Consortium. *Web Services Glossary*, February 2004. [Note] http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/.

[W3C04j] World Wide Web Consortium. *XML-binary Optimized Packaging*, August 2004. [Candidate Recommendation] http://www.w3.org/TR/2004/CR-xop10-20040826/.

[W3C04k] World Wide Web Consortium. *XML Information Set*, 2nd edition, February 2004. [Recommendation] http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.

[Wel00] Brian Wellington. *RFC 3007: Secure Domain Name System (DNS) Dynamic Update*. Internet Engineering Task Force, November 2000. http://www.ietf.org/rfc/rfc3007.txt.

[WGZP04] Xiaohang Wang, Tao Gu, Daqing Zhang, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. In *Workshop on Context Modeling and Reasoning at IEEE International Conference on Pervasive Computing and Communication*, March 2004.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.

[Win99] Dave Winer. *XML-RPC Specification*, October 1999. http://www.xmlrpc.com/spec.

[Woo01] David Woodhouse. JFFS: The journaling flash file system. Presented at the Ottawa Linux Symposium, October 2001. http://sources.redhat.com/jffs2-html/.

[WV02a] The Wireless Village initiative. *Wireless Village Specification, version 1.0*, March 2002.

[WV02b] The Wireless Village initiative. *Wireless Village Specification, version 1.1*, July 2002.

[YJWN02] Jukka Ylitalo, Petri Jokela, Jorma Wall, and Pekka Nikander. *End-point identifiers in Secure Multihomed Mobility*, December 2002. http://www.hut.fi/~jylitalo/publications/Opodis02-Ylitalo-et-al.pdf.

[ZFD+03] Youyong Zou, Tim Finin, Li Ding, Harry Chen, and Rong Pan. Using semantic web technology in multi-agent systems: a case study in the TAGA trading agent environment. In *Proceedings of the 5th international conference on Electronic commerce*, pages 95–101, September 2003.

[ZKJ01] Ben Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report CSD-01-1141, Computer Science Division, University of California, Berkeley, California, April 2001.