UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING, SCIENCE & MATHEMATICS

School of Electronics & Computer Science

Enhancing Distributed Real-Time Collaboration with Automatic Semantic Annotation

by

Benjamin Paul Juby

Thesis for the degree of Doctor of Philosophy

February 2005

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE & MATHEMATICS

School of Electronics & Computer Science

Doctor of Philosophy

Enhancing Distributed Real-Time Collaboration with Automatic Semantic Annotation

by Benjamin Paul Juby

Distributed real-time collaboration, such as group-to-group videoconferencing, is becoming increasingly popular. However, this form of collaboration tends to be less effective than co-located interactions and there is a significant body of research that has sought to improve the collaboration technology through a variety of methods. Some of this research has focused on adding annotations that explicitly represent events that take place during the course of a collaboration session. While this approach shows promise, existing work has in general lacked high-level semantics, which limits the scope for automated processing of these annotations. Furthermore, the systems tend not to work in real-time and therefore only provide benefit during the replay of recorded sessions. The systems also often require significant effort from the session participants to create the annotations.

This thesis presents a general-purpose framework and proof of concept implementation for the automated, real-time annotation of live collaboration sessions. It uses technologies from the Semantic Web to introduce machine-processable semantics. This enables inference to be used to automatically generate annotations by inferring high-level events from basic events captured during collaboration sessions. Furthermore, the semantic approach allows the framework to support a high level of interoperability, reuse and extensibility. The real-time nature of the framework means that the annotations can be displayed to meeting participants during a live session, which means that they can directly be of benefit during the session as well as being archived for later indexing and replay of a session recording.

The semantic annotations are authored in RDF (Resource Description Framework) and are compliant to an OWL (Web Ontology Language) ontology. Both these languages are World Wide Web Consortium (W3C) recommendations. The framework uses rule-based inference combined with knowledge from an external triplestore to generate the annotations. A shared buffer called a *tuple space* is used for sharing these annotations between distributed sites.

The proof of concept implementation uses existing Access Grid videoconferencing technology as an example application domain, to which speaker identification and participant tracking are added as examples of semantic annotations.

Table of Contents

Table of Figures

Acknowledgements

I would like to thank my supervisor, Dave De Roure, who has provided me with much support and advice throughout the duration of the work described in this thesis.

I would also like to thank Terry Hodgkinson and Stewart Fallis from BTexact for the helpful comments they have provided for this work. I am also grateful to BT for supporting me with a CASE award for the first three years of my research.

The members of the Pervasive Computing and Networks research theme at Southampton also deserve thanks for letting me bounce my ideas off them during the course of my research. I am particularly grateful to Danius Michaelides, Mark Weal, Ian Millard, Sajay Vivek and Xiang Fei for sparing the time to proof read and comment on the initial draft of this thesis.

I would finally like to thank my parents for their support throughout this work.

1 Introduction

Real-time collaboration that involves geographically distributed people is becoming increasingly commonplace. This is largely due to the now widespread availability of computers and networks able to handle multimedia data. Any person equipped with a standard PC, webcam and network connection can join a videoconference and collaborate with other people. Even sophisticated room based videoconferencing systems such as the Access Grid [Acc04] are now affordable by an increasingly large number of organisations.

Unfortunately, meeting using a remote collaboration technology such as videoconferencing can be less effective than meeting face-to-face. Hollan and Stornetta [Hol92] have proposed that by using computers to enhance distributed real-time collaboration, the potential exists to improve collaboration to the point that it becomes as effective as, or even *more* effective than non-mediated face-to-face collaboration.

There is a need to enhance distributed real-time collaboration to go beyond audio, video and simple data sharing. This thesis discusses automatic live semantic annotation as a way to enhance real-time distributed collaboration, focusing on group-to-group videoconferencing as a deployment scenario. The emphasis of the work is on the infrastructure required to generate and share the semantic annotations in real-time. It presents a generic framework based around Semantic Web technologies, and this framework is demonstrated with a proof of concept implementation.

Semantic annotation in this context means giving the individual events that occur as part of a collaboration activity an explicit representation that has a formally defined meaning. In practical terms this means that in addition to distributing audio and video streams between sites in a videoconference, real-time generated descriptions of the events in the session are also distributed between sites. Examples of useful annotations are agenda items, speaker identification and tracking when participants join or leave the session. These semantic annotations that represent events can be displayed to session participants in real-time.

Displaying annotations in real-time during a live session has a number of potential benefits. In particular, annotations can provide useful additional information that would otherwise only be implicit or only available to some participants. This could potentially help increase the level of situation awareness of the participants and lead to more effective collaboration. Furthermore, the semantic annotations may be archived in addition to audio and video to provide an index for the recording of the session. The potential benefits of such an index include facilitating the navigation and searching of recordings, allowing higher level queries, and also for reusing recorded material. The annotations can also be replayed in synchronisation with the media streams to provide a more complete replay than audio and video alone could provide.

While there is existing work on annotation of real-time collaboration sessions, it has not used high-level semantics for describing meeting events, i.e. explicit machine-processable annotations combined with machine-processable semantics. This limits the amount of automated processing that can take place. Additionally the existing systems mostly do not work in real-time, thus only providing any benefit when replaying a recorded session. The systems also generally require significant effort from the participants to create the annotations and have focused on co-located collaboration, meaning they have poor support for distributed collaboration.

The semantic approach to annotation presented in this thesis has a number of advantages over non-semantic approaches. In particular it allows inference to be used, which is applied to enable high-level events to be automatically derived from basic events captured in collaboration sessions. A semantic approach also enables a high level of interoperability, reuse and extensibility.

The framework presented here uses rule-based inference combined with knowledge obtained from an external repository called a triplestore. A shared buffer known as a tuple space is also used for sharing the annotations between distributed sites

The semantic annotations are authored using the Resource Description Framework (RDF), which is a language for representing information about resources in the World Wide Web. The vocabulary used for annotation is specified using the Web Ontology Language (OWL). This language is used to define vocabulary terms, their meanings and their interrelationships.

The proof of concept implementation uses Access Grid videoconferencing technology as an example application domain, to which speaker identification and participant tracking are added as examples of semantic annotations. Although Access Grid is used as the example domain, the implemented system is not dependent on Access Grid technology and is general purpose enough to be used with other room-based conferencing technologies.

1.1 Contributions

This thesis presents a novel investigation into the application of knowledge technologies to computer mediated collaborative applications. It therefore contributes research to the areas of Computer Supported Cooperative Work (CSCW) and the automated creation of content for the Semantic Web. The core contribution can be summarised as follows:

The application of Semantic Web technologies to the domain of real-time distributed collaboration. Existing uses of Semantic Web technologies have been largely limited to non real-time domains. This thesis demonstrates their use in a live, real-time domain, and additionally demonstrates the use of a real-time tuple space to distribute semantic annotations between non co-located collaborating users.

Existing systems for semantic annotation rely heavily on hand authoring of information, which places significant burden on users. The rule-based inference approach demonstrated here almost completely automates the process of live semantic annotation and requires almost no additional effort from the day-to-day users of the system.

The automated real-time approach to the generation of the semantic annotations enables the novel functionality of being able to display the annotations to the session participants in real-time as soon as they are generated. This means that in addition to the more traditional use for indexing and replay after the session, the annotations can directly benefit the participants during the session.

This thesis also identifies a number of useful meeting events that are common to many different meeting types. Some of these events are formalised into an ontology for describing live collaboration sessions. This live collaboration ontology demonstrates the powerful features of the Semantic Web for reuse by reusing a number of existing ontologies to create certain parts of the live collaboration ontology.

1.2 Document Structure

This thesis is arranged as follows:

Chapter 2 reviews a selection of relevant literature, including existing computer enhanced collaboration systems and the Semantic Web.

Chapter 3 explains and motivates the use of semantic annotations for enhancing distributed real-time collaboration.

Chapter 4 presents an event based framework for the automatic semantic annotation of distributed real-time collaboration activities.

Chapter 5 describes a proof of concept implementation of the framework discussed in chapter 4.

Chapter 6 describes in detail the inference process that the proof of concept system used.

Chapter 7 presents a discussion based analysis of the system framework and implementation.

Chapter 8 discusses the conclusions for the thesis.

2 Literature Review

This chapter reviews a selection of literature relevant to the research in this thesis and is divided into two main parts after the introduction. The first part looks at a number of existing computer enhanced collaboration systems and reviews them according to a number of relevant criteria. The second part covers the Semantic Web, starting by looking at the main specifications used and then discussing a number of Semantic Web applications, some of which are used for collaborative purposes.

2.1 Introduction

The research in this thesis is within the domain of Computer Supported Cooperative Work (CSCW), which is "a generic term which combines the understanding of the nature of group working with the enabling technologies of computer networking, systems support and applications" [Rod91].

Group work is commonly classified into spatial relationships between workers and their temporal relationships [Rod91]. Collaboration can either be local or remote and synchronous or asynchronous. Synchronous collaboration involves people interacting in real-time, whereas in asynchronous collaboration the interactions are non real-time and do not require an immediate response. For example, asynchronous remote collaboration often uses e-mail or the web, and synchronous remote collaboration may use a telephone or videoconference. Local synchronous collaboration may involve meeting support tools and local asynchronous collaboration may involve shared document authoring. Often a combination of these modalities will be used during the lifetime of the collaboration activity.

2.1.1 Early Work

Some of the earliest work in CSCW was pioneered by Douglas Engelbart [Eng62]. His work identified that computers could be used as a communication mechanism for team cooperation, allowing people to work simultaneously on the same materials. He hypothesised that this would lead to a significant increase in group problem solving ability. Although he provided no firm evidence to prove this hypothesis, he did however claim that from his own personal experience he had noticed a "really phenomenal boost in group effectiveness over any previous form of cooperation" [Eng62] as a result of using computer based collaboration tools.

Later work by Engelbart [Eng75] harnessed the ARPANET combined with telephone audio conferencing for computer augmented real-time distributed group collaboration. The system used a shared display allowing remote people to view and control the same computer display, enabling them to access notes and working records, copy materials and access shared whiteboard functionality.

Other notable early work in this domain was carried out by Hiltz and Turoff [Hil81]. Their Electronic Information Exchange System (EIES) provided features for synchronous and asynchronous text based group conferencing. It provided a directory for locating other users and supported features such as voting and shared notebooks. It also provided archiving and indexing of discussion, which allowed searching by topic, author or date.

2.1.2 Colab and Media Spaces

By the mid 1980s research in CSCW had started to take off in a big way. One of the key players at this time was the Xerox Palo Alto Research Centre (PARC) [Goo86]. The centre's notable research contributions included the Colab and Media Spaces projects. The Colab [Ste86] was a computer enhanced meeting room (see figure 2.1) in which each participant had access to a networked computer allowing them to structure and share meeting information through a multi-user interface called WYSIWIS (What You See Is What I See). The room was also equipped with a full size touch sensitive digital whiteboard called a Liveboard. Not only did the Colab allow the structuring and shared manipulation of meeting artefacts, but removed the need to transcribe these to the participants personal computers after the meeting.

At around the same time as the Colab project, work was underway on the Media Spaces project (described in a later paper by Bly *et al.* [Bly93]). This linked the offices and communal spaces of two sites separated by several hundred miles with always on audio and video connections. This provided the kind of informal contact between the distributed workers that collocated workers can take for granted. In addition to traditional videoconferencing, it provided peripheral awareness, giving an overview of who was around and what was happening, and allowed chance encounters and social

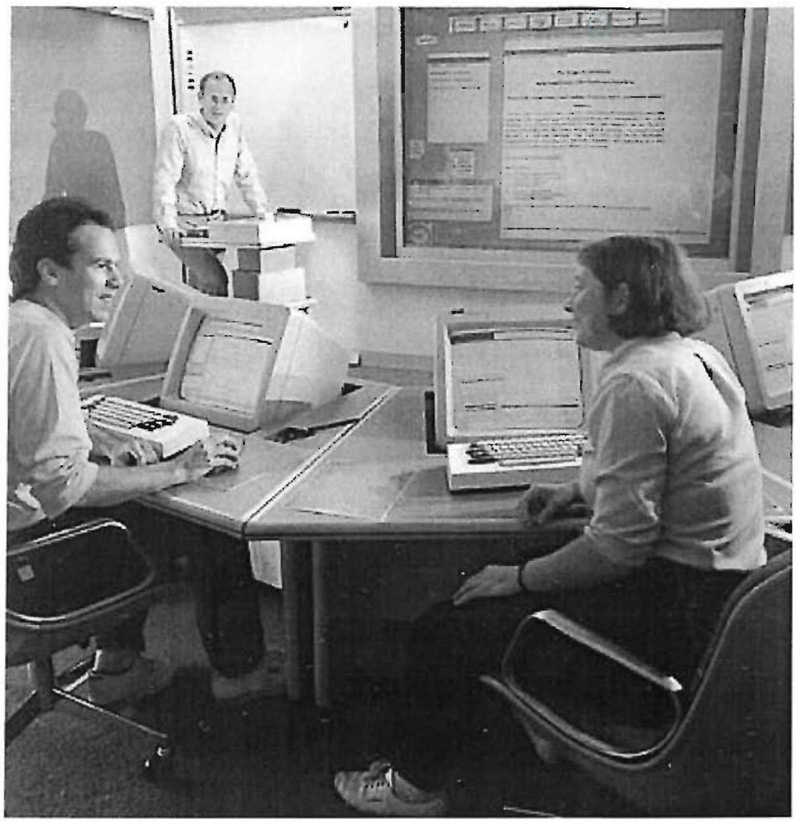


Figure 2.1, The original Colab in use at Xerox PARC (from [Ste87])

activities. A follow up project at EuroPARC called Portholes [Dou92] used less heavyweight techniques to provide periodic video snapshots from individual offices to give a general sense of awareness of who was around and what they were up to.

2.1.3 Videoconferencing Interfaces

Work such as that of Buxton *et al.* [Bux97] has looked at provide improved interfaces for videoconferencing to try and address weaknesses such as a lack of eye contact, failure to perceive the group as a whole and inability to hold side conversations. Work on this task lead to the development of the Hydra system [Sel92], which simulated a four way round-table meeting, with one physical participant and up to three remote participants represented by their own “video surrogate” unit consisting of a camera, monitor and speaker (see figure 2.2). Since each participant occupied a distinct place around the meeting table, this preserved gaze and head turning and allowed side conversations. An evaluation of the system showed that although the structure of turn taking behaviour was not found to be significantly different when compared to regular videoconferencing, it did support parallel and side conversations, which the regular videoconferencing system did not.

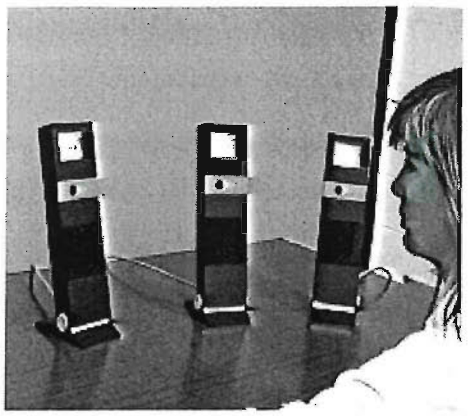
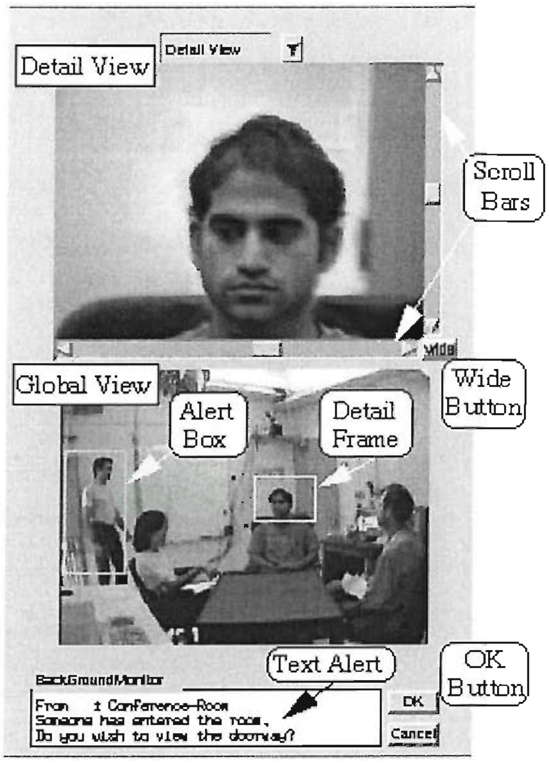


Figure 2.2, the Hydra system in use showing three video surrogates (from [Sel92]).

Before Hydra, other researchers had provided alternative forms of support for gaze awareness. For example “video tunnels” described by Acker and Levitt [Ack87] were videoconferencing terminals that used a half silvered mirror at 45° to reflect an image of the user into a camera on top of a monitor. The mirror allowed the camera to effectively point directly into the eyes of the person looking into the monitor, thus accurately conveying gaze information to the remote users.

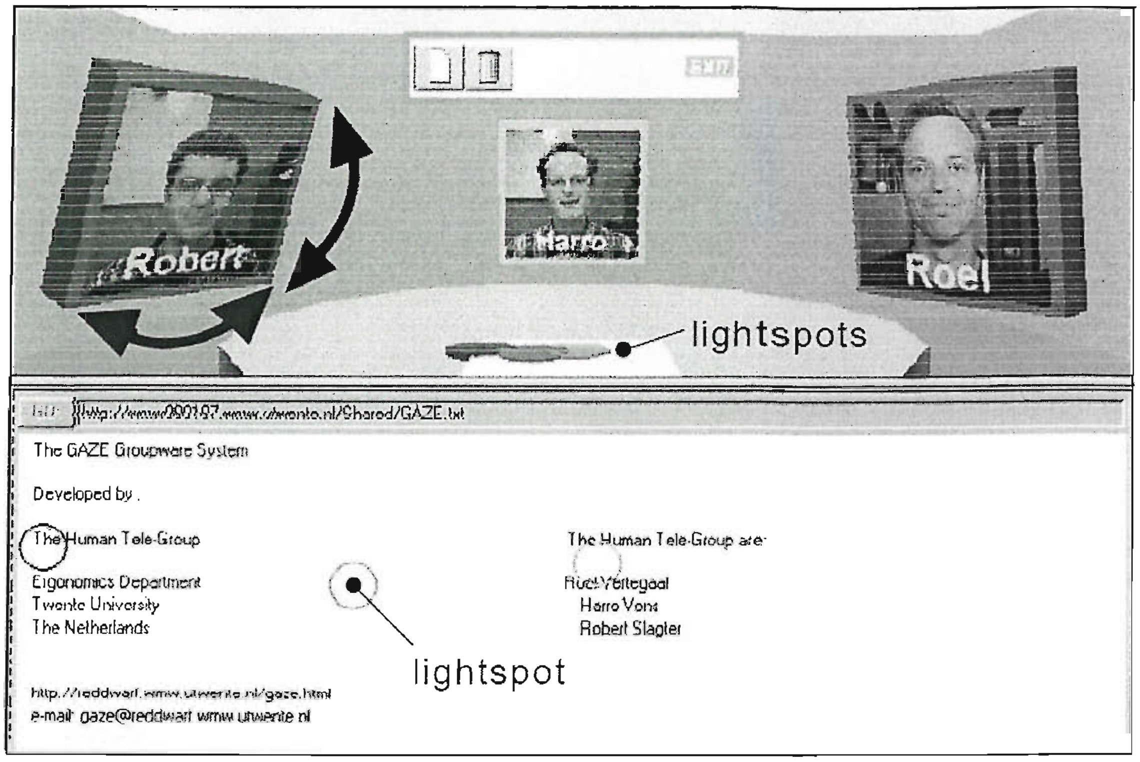
Other research led by Buxton resulted in the Extra Eyes videoconferencing system [Yam96], which was designed to compensate for the lack of peripheral awareness during videoconferences. When using existing systems, tasks such as keeping track of who is at the remote site or what they are doing can be difficult as the view is limited to whatever is in front of the camera. Extra Eyes provided a peripheral, wide-angle global view simultaneously with a close up detail view (see figure 2.3). A bounding box displayed in the global view precisely identified the region that was displayed in the detailed view. Clicking in the global view controlled the remote detail camera and caused it to be oriented to point at that new position. This interface made the relationship between the global and detail views explicit and reduced the potential for any confusing spatial discontinuities. Sensors in the room also detected the entry of new participants and flagged this with an alert box in the global view and a message asking if the viewer would like to view the doorway. An evaluation was conducted that involved identifying different letters displayed on video monitors at changing positions in the remote room. The letters were too small to read in the global view, requiring the user to move between the monitors with the detailed view. The evaluation showed that the linking between the global and detailed views made the task completion time significantly quicker and that the addition of explicit alerts showing when a letter changed (and its new position) further sped up the task completion.



From : Conference-Room
Someone has entered the room.
Do you wish to view the doorway?

Figure 2.3, The Extra Eyes user interface (from [Yam96])

One of the main drawbacks of systems like Hydra, video tunnels and Extra Eyes was the specialist hardware required and the complexity of the set up. Vertegaal *et al.* [Ver98] developed the GAZE Groupware system to provide a more lightweight method of maintaining group awareness and communicating gaze information. The system used a PC based eye tracking camera that conveyed gaze information in a shared three-dimensional virtual meeting room. Each participant is represented as a portrait based personification around the table in the virtual meeting room (see figure 2.4). Each personification rotates according to where the corresponding participant looks. For example, if person A looks at person B, then B sees A's personification turn to face them. When A looks at person C, then B sees A's personification turn towards C. Furthermore, when a participant looks at a document in the shared workspace, a virtual "lightspot" is projected on the document indicating which part of the document they are currently looking at. The colour of the lightspot corresponds to the colour of the border around the participant's personification.



http://www090197.www.utwente.nl/Shared/GAZE.txt

The GAZE Groupware System

Developed by .

The Human Tele-Group
Ergonomics Department
Twente University
The Netherlands

The Human Tele-Group are:
Roel Vertegaal
Harro Vons
Robert Slagter

http://reddwarf.wmw.utwente.nl/gaze.html
e-mail: gaze@reddwarf.wmw.utwente.nl

Figure 2.4, The interface to the GAZE system (from [Ver98]).

2.1.4 Conversational and Workspace Awareness

The novel videoconferencing interfaces discussed in the previous section were created to enhance videoconferencing through improving the level of conversational awareness and workspace awareness amongst the participants.

Conversational awareness [Ver97] is awareness about what is happening in a conversation. It provides information about who is communicating with whom and "answers both mechanical questions (did they hear me, did they understand me, who's going to talk next?) and also affective questions (do they believe me, how are they reacting?)" [Gut97]. This awareness comes from cues such as eye contact, gestures and intonation.

Workspace awareness is the maintenance of awareness "about others' locations, activities and intentions relative to the task and the space" [Gut96]. Gutwin [Gut97] has identified the categories of knowledge that make up workspace awareness and the specific elements within those categories. These are summarised in Table 2.1, along with a list of specific questions that each element answers. Its possible to see that there is some amount of overlap between workspace and conversational awareness, especially with the presence, identity, authorship and gaze elements of workspace awareness,

While conversational and workspace awareness usually comes naturally in face-to-face interactions, in videoconferencing this awareness has to be explicitly designed into the systems, as has been shown in the previous sections. The Hydra system mainly focused on conversational awareness, Extra Eyes mainly focused on workspace awareness (by providing peripheral awareness, which is a subset of workspace awareness) and the GAZE system provided both conversational and workspace awareness.

Category	**Element**	**Specific questions**
Who	Presence	Is anyone in the workspace?
	Identity	Who is participating? Who is that?
	Authorship	Who is doing that?
What	Action	What are they doing?
	Intention	What goal is that action part of?
	Artefact	What object are they working on?
Where	Location	Where are they working?
	Gaze	Where are they looking?
	View	Where can they see?
	Reach	Where can they reach?

Table 2.1, Elements of workspace awareness (from [Gut97]).

2.1.5 Computationally Mediated Interactions

The integration of people, pervasive computation and physical reality (such as the Colab described in section 2.1.2) is sometimes referred to as a *smart space*. Mark [Mar99] describes a long-term vision for a special kind of smart space called a *mediated space*. In a smart space, humans deal directly with computational devices to accomplish task. In a mediated space, individuals primarily interact with each other and not with the space (although explicit interaction with the space may still occur). The mediated space improves human activities by enhancing the interaction of people in the space by proactively suggesting relevant information from outside the space and providing other features such as checking the consistency of interactions with previous interactions. This is achieved through the space understanding the interactions taking place, using techniques such as speech and gesture recognition. Mark predicts that technology to achieve this will emerge over the course of the next 15 years.

While Mark describes computational mediation for co-located people, Hollan and Stornetta [Hol92] describe the potential for mediated communication for people who are not co-located. They argue that the potential exists to improve computationally mediated communications to the point that it becomes as effective as, or even more effective than non-mediated face-to-face communication. They identify a number of specific advantages that mediated communications could have. These are summarised here:

- **Clarity**. Natural spoken language can be imprecise and ambiguous. Through spatial location of the objects of discussion in a shared visual space, specific objects could potentially be referred to by pointing at them. This would, for example, overcome the reference ambiguity of using the word "it" in an English sentence.

- **Feedback**. Facial expressions and verbal cues are used to indicate to a speaker that their conversation is being followed. It is argued that these mechanisms are rather imprecise. For example, the speaker may be unclear as to what aspects of what they are saying the listener understands or what the listener thinks their key point is. With spatial location of key pieces of the discussion in a shared visual space, the listener may be able to provide a rich range of feedback that simultaneously indicates what aspect of the speaker's comments they are responding to.

- **Archiving**. A searchable audio and visual record of the interaction could potentially be created automatically.

Hollan and Stornetta use the term "auditory paper" to describe this proposed real-time visual extension to natural language. They also identify that unlike face-to-face interactions, computationally mediated communication may be asynchronous, which removes the need for all parties to be free at the same time, and hence promotes interaction.

2.2 Review of Computer Enhanced Collaboration Systems

This section reviews six existing research systems that provide some form of computer mediated enhanced support for the annotation or capture of primarily synchronous collaboration activities, such as meetings or videoconferences. The list of systems reviewed here is not exhaustive, as other similar systems do exist. These were however chosen as the major systems that have implemented concepts which are relevant to this research.

2.2.1 Review Criteria

Each system will be introduced with a summary of its functionality and will then be reviewed according to a number of different criteria. The overall goal of the review is to show how these existing systems compare to the new approach presented in this thesis.

The first two criteria are intended to expose the precise problem space that each system tackles. This shows how general purpose the approach is and how it relates to the problem space of this thesis (i.e. supporting live temporal annotation of distributed meetings, and the potential for indexing and replay of those meetings). The specific criteria are:

- **Type of collaboration supported**. This considers factors such as if the system supports useful features like distributed collaboration or any specific scenarios such as lectures or meetings.

- **Type of information added**. This considers what the system provides over traditional videoconferencing or video recordings, and in particular, the types of annotations or events it is able to capture and represent.

The remaining criteria examine to what extent the systems support the key desirable features of the approach presented in this thesis; these features being support for machine processable annotations and semantics, live processing, and automation. Hence the following criteria were established:

- **Support for machine processable annotations and semantics**. This considers the scope for automated processing of the information added by the system and if there are any machine processable semantics associated with this information.

- **Support for live processing**. This considers which features the system provides during the collaboration session and which features only become available after the session.

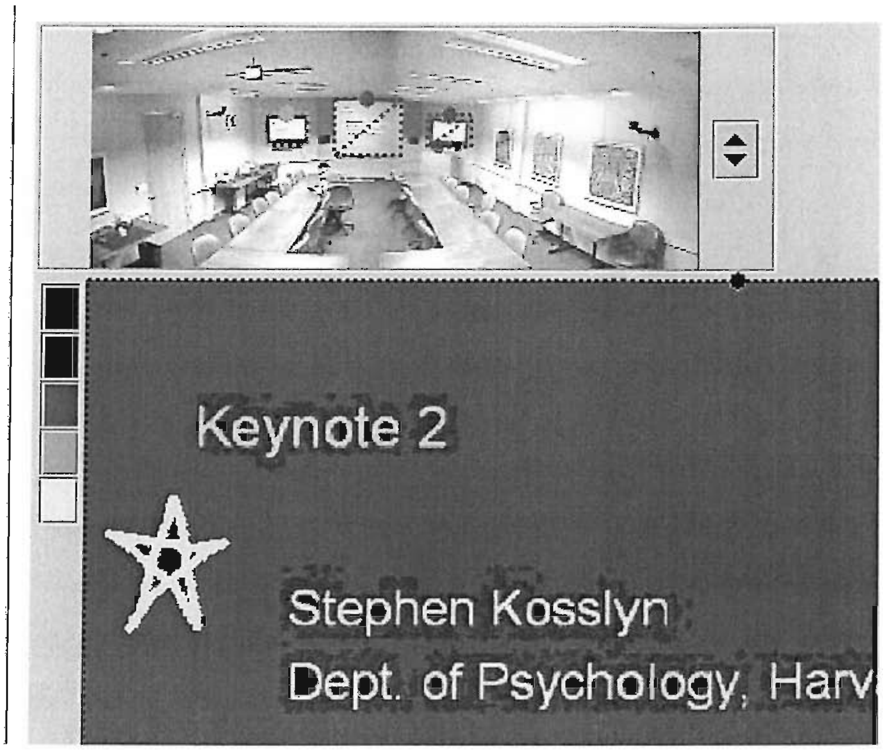
- **Degree of automation**. This considers the amount of human input required by the system before, during or after the collaboration session.

2.2.2 NoteLook

NoteLook [Chi99b] is a system for indexing and annotating meetings that take place in a specially constructed meeting room at FX Palo Alto Laboratories [Chi99a]. The room is equipped with video cameras, microphones, video projectors and a wireless network. The system uses tablet PCs to capture freehand notes taken by meeting participants and the wireless network enables presentation slides displayed on the video projector to be automatically displayed on the tablet PCs in real-time. This enables meeting participants to annotate directly over the slides using digital ink as the meeting progresses. A sequence of thumbnails from the room cameras is also added to the note pages. This can help determine who was speaking at that time and what was going on. These notes are time-stamped and correlated to the multimedia data.

Version 3.0 of NoteLook [Chi03] also adds a feature allowing the hand authored notes to be displayed to other meeting participants during a live meeting. As shown in figure 2.5, users of NoteLook 3.0 are presented with a live panoramic view of the meeting room on their tablet PCs on to which is overlaid an augmented reality interface, which allows users to 'drag' a slide off one of the wall displays, annotate it and then 'drag' it on to another wall display for immediate display to the other participants.

At the end of a meeting, the system then can be instructed to automatically generate a web-based index and archive page for the meeting. This consists of a miniature version of each note page displayed on the index page, which can be enlarged by clicking on them. This is shown in figure 2.6. Clicking on a note page or an individual freehand annotation will start the playback of the meeting at the point at which that note was taken. This takes the form of synchronised replay of the recorded video and the annotated slides, played back through the NoteLook system.


Keynote 2

Stephen Kosslyn
Dept. of Psychology, Harv

Figure 2.5, The Notelook 3.0 freehand annotation interface (from [Chi03])

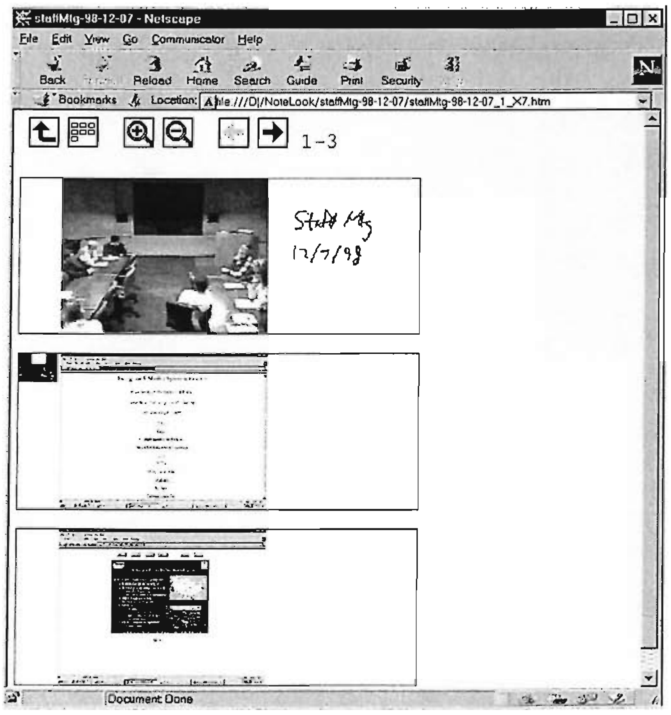

staffMtg-98-12-07 - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///D|/NoteLook/staffMtg-98-12-07/staffMtg-98-12-07_1_X7.htm
1-3
Staff Mtg
12/7/98
Document: Done

Figure 2.6, The Web based NoteLook index (from [Chi99b])

Type of Collaboration Supported

The system supports co-located participants in a meeting scenario that uses presentation materials. Chiu *et al.* discuss the possibility of using the system in videoconferences, but have not implemented this. The system also archives the slides, annotations and meeting video for later viewing.

Type of Information Added

The system captures presentation materials, freehand annotations and video thumbnails, which are all timestamped. The system also allows live display of annotations to other participants. This captured information is used to automatically generate a web-based index and archive of the session.

Support for Machine Processable Annotations and Semantics

Apart from an explicit notion of slide display events, the system has no real machine processable information. For example, annotations are left as freehand pen strokes and speaker identification is achieved by a human viewing the video thumbnails on a note page.

Support for Live Processing

Annotations are taken during live meetings, and can be displayed to other participants as soon as they have been authored. However, the index pages aren't generated until the end of a meeting.

Degree of Automation

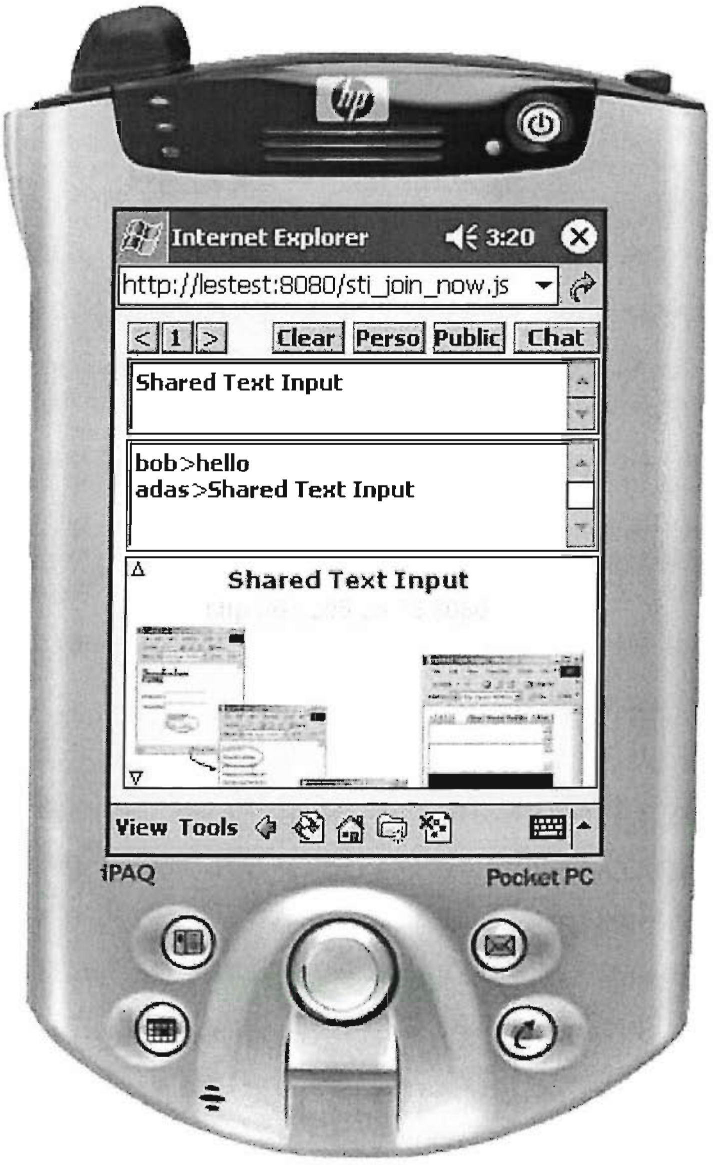
The system can automatically provide participants with the current slide and a number of video thumbnails which to annotate over, but the participants still have to write all the annotations manually. The system does however automatically generate a meeting index and record from the captured slides, video and annotations.

2.2.2.1 Conclusions about NoteLook

The presentation material centric approach of the NoteLook system means that its usefulness is fairly limited in meeting scenarios that don't use slides. While the automatically generated meeting index and record pages appear to be useful, the system's lack of semantics means that the information it captures is only suitable for human consumption, which severely limits the potential for further automated processing. The use of freehand notes makes its annotation capability very flexible, but does require significant manual input from participants.

2.2.3 Shared Text Input

Shared Text Input [Den04] is a PDA-based system designed to allow students to author and share notes in real-time during lectures using wireless PDAs or laptop PCs. If presentation slides are used, these are also displayed on the PDAs synchronised with the slide transitions from the lecturer. A picture of the system running on a PDA is shown in figure 2.7.



Shared Text Input

bob>hello
adas>Shared Text Input

Shared Text Input

Figure 2.7, The Shared Text Input application running on a PDA (from [Den04])

To enable faster note taking on the PDAs, students may reuse words from other notes or presentation materials simply by selecting the word. As well as promoting faster note taking, it is suggested that sharing the notes in real-time increases the awareness of the students during the lecture. After a lecture, the system automatically archives the notes and any presentation slides and places them on the web for future reference by students.

In a user trial of the system, some of the notes were found to contain URLs, allowing students to look at additional references during the class. The system was also used as a real-time chat tool, allowing students to ask each other questions during a lecture.

Type of Collaboration Supported

The scenario the system is designed for is that of a lecture, but the system could also be applied to a more general meeting room scenario. The notes and presentation materials are archived enabling viewing after the lecture.

Type of Information Added

The system allows hand authored text notes to be shared in real-time during a lecture and archived notes to be made available after the lecture.

Support for Machine Processable Annotations and Semantics

The system has some basic semantics as it supports the authoring of three different types of notes. Private notes that are not shared, chat notes that have a username appended to them so that they can be attributed to a particular student, and public notes, which do not have a username added. These different types are of limited use, as they only indicate how the note should be distributed and displayed. Some potential does exist for automated processing of the text, as the notes are in full text rather than in native handwriting.

Support for Live Processing

The system has good support for live processing; notes may be authored and shared in real-time during a lecture.

Degree of Automation

The system requires manual input of notes, but does allow note takers to re-use words from other notes. The system also automatically places the notes and any presentation materials on the web after the lecture.

2.2.3.1 Conclusions about Shared Text Input

Shared Text Input has shown a novel interface for reuse of other people's words to allow faster note taking during lectures. It has also been suggested that sharing notes in real-time can help improve the awareness of the lecture audience.

Through real use it has been shown that as well as proving a note taking facility, the back channel communication features are useful for things such as sharing URLs and asking questions. Some users however found that seeing live shared notes from all users caused them to experience 'information overload', so this shows that it is important to limit the amount of information displayed to users at once.

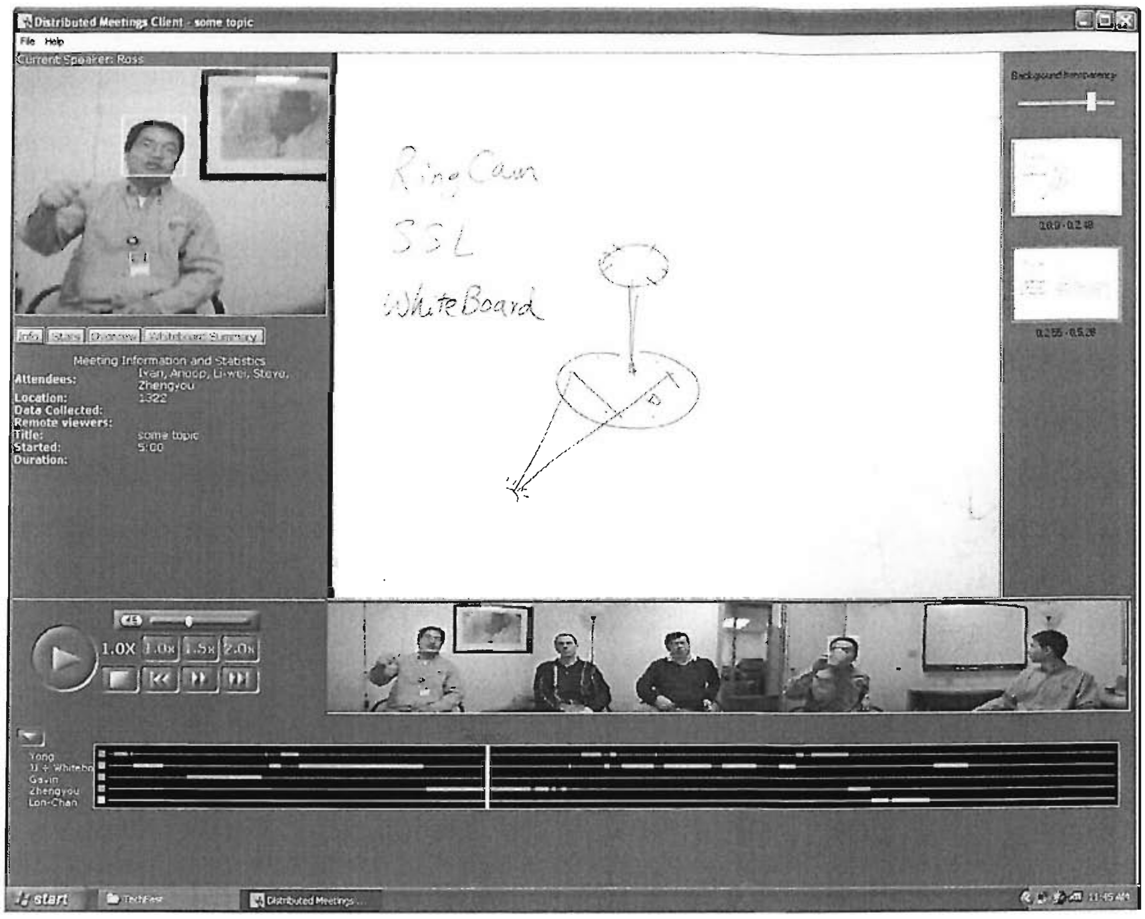
2.2.4 Distributed Meetings

The Distributed Meetings system [Cut02] enables the live broadcasting and recording of meetings. The system automatically captures a significant amount of extra information compared to a traditional audio or video recording.

The system uses a 360° panoramic video camera in the centre of the meeting table to capture and broadcast a view of all the participants. The system has a microphone array, which it uses to perform sound source localisation (SSL) on people speaking. This is combined with computer vision-based participant tracking techniques to determine where in the panoramic video the current speaker is. This is used to show a close up view of the current speaker obtained from the panoramic video. An additional camera also captures any whiteboard activity. The video streams are broadcast live to remote clients using multicast. Remote audio communication is achieved using the standard public telephone system.

All this information is also recorded and archived for later replay and a kiosk in the meeting room allows participants to start and stop the recording. The participant list and meeting description can be automatically obtained from the initial requests to hold a meeting using a Microsoft Exchange server. If additional participants are present who were not included in that request, they may be specified by holding a smart ID card next to a reader at the kiosk.

The archived meetings are automatically indexed by a speaker timeline and whiteboard pen strokes. The archived meeting client is shown in figure 2.8.



Meeting Information and Statistics

Attendees: Ivan, Anoop, Li-wei, Steve, Zhengyou
Location: 1322
Data Collected:
Remote viewers:
Title: some topic
Started: 5:00
Duration:

Figure 2.8, Distributed Meetings archived meeting client (from [Cut02])

Clicking on the speaker time line or a pen stroke starts the replay of meeting from that point. As the system is unable to identify specific participants in the meeting room, the name of each person in the timeline must be entered by hand as part of the archiving process. The system also has a time compression feature for replaying the meeting speeded up, enabling the meeting to be watched in less time.

As part of its evaluation, the system was used to record ten real-life team meetings where one or more team members were unable to attend the meetings. After the meetings the absent team members then watched the recordings and were asked to fill in a questionnaire. This presented the users with a number of statements about the usefulness of the system and they were asked to rate to what extent they agreed or disagreed with each statement. For example, one of the statements was "Being able to browse the meeting using the timeline was useful". There we similar statements about the usefulness of the time compression, panoramic view and speaker view features. The survey results showed that on average the users agreed with the statements about these features. As general comments, some users suggested that if the meeting was full of strangers, they would find the names on the speaker time line especially helpful and one user suggested that it may be useful to have other meeting events marked on the timeline also.

Overall the evaluation showed that the users at least perceived these features to be of benefit, although it did not present any conclusive evidence of time savings or improvements in understanding of the recordings by the users when compared to traditional meeting records such video recordings or minutes.

Type of Collaboration Supported

The system is intended to support a meeting scenario where participants are co-located in a single meeting room, and other participants may join the meeting over the telephone and watch the video streams from the meeting room. Participants unable to attend the meeting or watch the live broadcast can later watch a recording of the session.

Type of Information Added

In a live meeting, the features provided are limited to useful camera views. These are panoramic video, close ups of the current speaker and the whiteboard. In a recorded session, several types of meeting metadata are available in addition to the novel camera views. These additions are the meeting details (time, location, duration, title, names of participants, who led the meeting, number of active participants) and indexing using a speaker time line or whiteboard pen strokes.

Support for Machine Processable Annotations and Semantics

The system has an explicit representation of some basic meeting metadata, such as the meeting details, individual participants speaking and whiteboard pen strokes.

Support for Live Processing

The system supports live broadcast of its various video views, but the indexing functionality is only available after the meeting has ended.

Degree of Automation

The camera control, speaker identification, speaker time line and index from whiteboard pen strokes are all created automatically. However, the names of the participants on the speaker time line have to be manually entered.

The meeting details are automatically obtained from the initial meeting set-up in Microsoft Exchange, although participants not included in the original communications to arrange the meeting have to manually enter their details in the meeting room kiosk.

2.2.4.1 Conclusions about Distributed Meetings

Distributed Meetings appears to be a useful tool for automatically capturing meeting metadata. However, its available features in a live meeting are fairly limited, as the system provides no beneficial features for participants located in the meeting room and simply provides some useful camera views for display the remote participants. Furthermore, remote participants are treated as second class citizens, as the video is only one way and their audio is not included in the speaker identification. They also have no way of drawing on the whiteboard.

Its features for archiving meetings seem to be more useful. A first hand user account mentioned that speaker identification was a useful way to index archived meetings, and identifying participants by name on time line would be useful if the participants were not known to the person viewing the replay. Another user also said that it would be useful to record other events, such as people leaving or joining the meeting room.

2.2.5 The AVIARY Intelligent Room

The AVIARY (Audio-Video Interactive Appliances, Rooms and sYstems) intelligent room testbed [Mik00, Hua03] is a system that handles the automated capture of multi-person interactions in a meeting room. The room is equipped with static cameras, active pan/tilt/zoom cameras and microphones. This allows the remote viewing of a live meeting or the later browsing and viewing of a recorded meeting. The capture takes the form of location tracking of participants and speaker identification, which is used to automatically control the cameras in the room to select the best shots and to build up a graphical summarisation of the meeting (shown in figure 2.9). This graphical summary is used to browse recorded meetings and locate sections of interest for video replay.

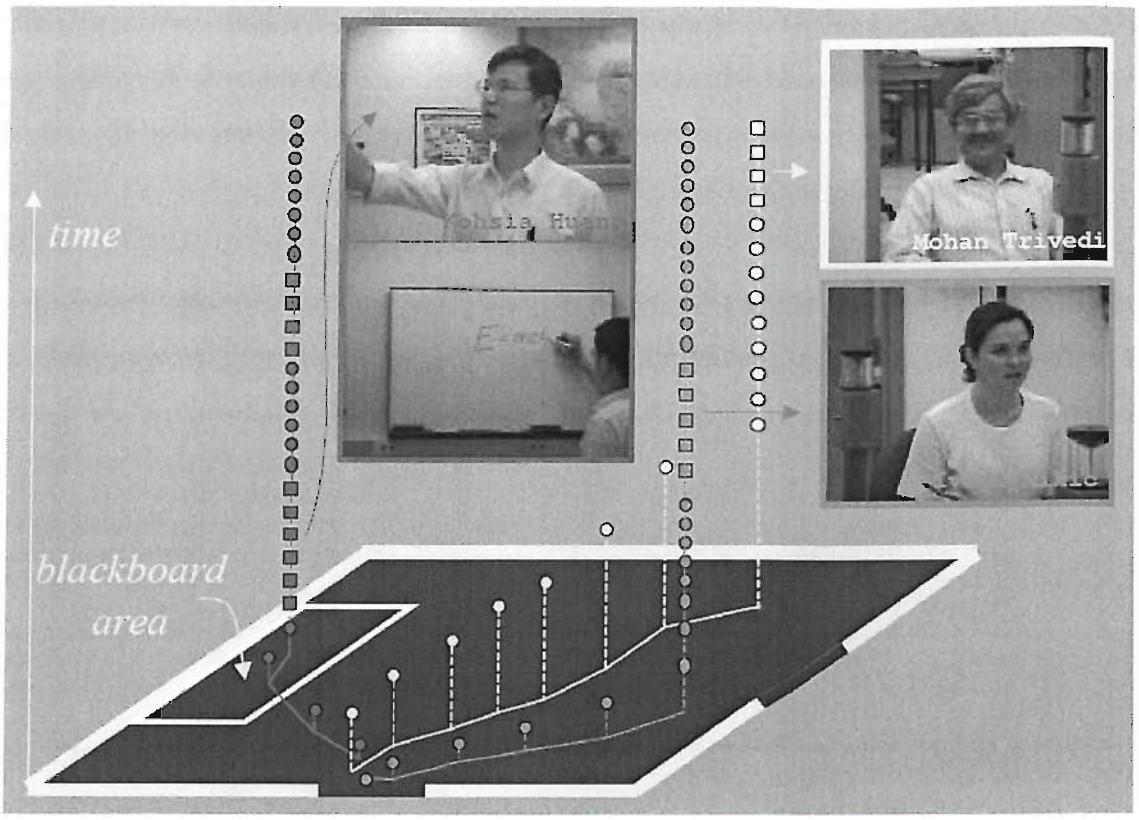


Figure 2.9, The AVIARY graphical summary (from [Mik00])

The system is able to recognise three different types of event in the room, these are: when somebody is located in front of the whiteboard, when that person (i.e. the lead presenter) speaks, and when other participants speak. The purpose of recognising these events is to be able to automatically direct the cameras to capture these events and also to mark them on the graphical summary.

In order to recognise these events, the system employs a number of computer vision techniques combined with voice recognition. When a person enters the room, they are required to speak in order that the system may use voice recognition to identify them and this is combined with face recognition to ensure reliable identification. The static cameras are then used to perform 3D tracking of each participant within the room so that the system knows the location of each participant at any moment in time. Whenever a participant speaks, voice recognition is used to automatically identify that participant, and the system uses this knowledge along with the location information to know where to aim one of the cameras in order to obtain a close up of the speaker. Similarly the system can use the location information to detect when a participant is using the whiteboard.

As well as automatically selecting optimum camera shots, the event and location information is used to generate a 3D graphical summary of the meeting, which is generated in real-time during the meeting and may be used to navigate through the recorded video after the session. The summary shows the room floor plan, with a third vertical axis representing time. Each participant's activity is represented as three dimensional track, which shows their location in the room over the duration of the meeting. This makes it possible to, for example, determine when a particular participant drew on the whiteboard. Along each track are multiple squares or circles, a square representing that the person was speaking and circle for when they were not speaking. When the user selects a specific track they are shown a face snapshot and the name of the person the track represents. Clicking on a square or circle begins the replay of the video from that point.

Type of Collaboration Supported

The system supports group meetings of co-located individuals with other remote people passively viewing the live session. It also supports the later browsing and replay of recorded meetings.

Type of Information Added

The system is able to keep track of who is in the room, where in the room they are and who is currently speaking. During a live meeting this information is used to automatically control the cameras and dynamically construct a graphical summary of the session. After a session, the graphical summary can be used as an index for the recorded video.

Support for Machine Processable Annotations and Semantics

AVIARY is able to recognise three interesting meeting events, and also uses a rudimentary form of inference to determines when these events are occurring. It achieves this by combining participant location data, speaker identification data and existing knowledge about the room.

Support for Live Processing

It has fairly good support for live processing as the event recognition and automatic camera control both work in near real time, although the speaker identification module requires 1-5 seconds of speech before identification may occur, so this introduces some delay. The graphical summarisation is also constructed in on-the-fly and can be displayed to local participants and remote viewers.

Degree of Automation

The camera control and generation of the graphical summary are both done automatically. The participants must however always remember to speak as soon as they enter the room to allow person identification.

2.2.5.1 Conclusions for The AVIARY Intelligent Room

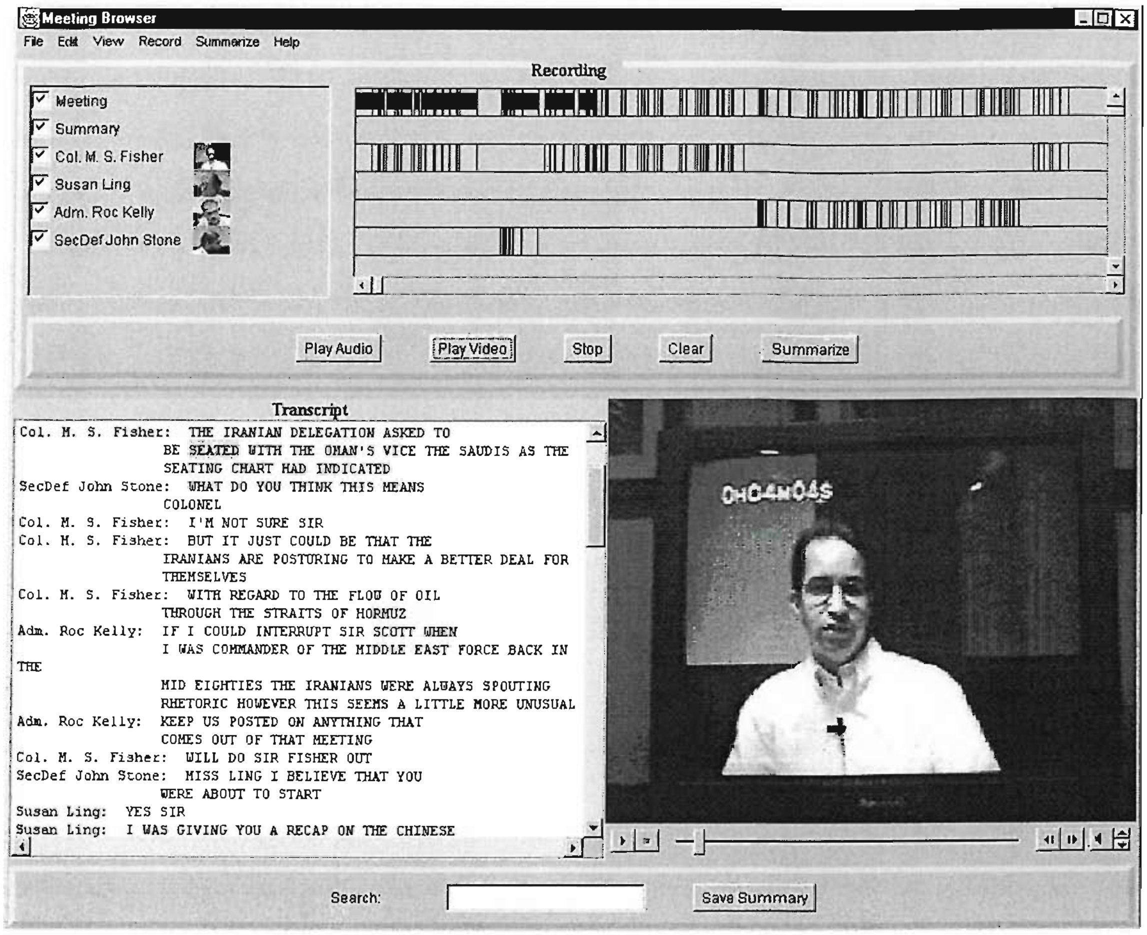
The most interesting feature of the AVIARY system is that it uses basic information to infer interesting meeting events, although it doesn't use a general purpose knowledge representation, so extending the system to perform other inferences would be difficult.

The relatively long time required for speaker identification means that the system takes significant time to respond (up to 5 seconds) and is likely to completely miss short utterances altogether. It also requires all participants to speak as soon as they enter the room, which could potentially be disruptive to a meeting if a participant joined the meeting after it had already started.

The three dimensional summarisation of meetings is a novel representation, but in some cases it may lead to a very complex representation that is difficult to understand, especially if the participants move around frequently during the meeting.

2.2.6 The Smart Meeting Room Task

The Smart Meeting Room Task (SMaRT) [Wai03] is a research activity with an overall goal to provide a smart meeting room that supports people in any kind of meeting situation, without any explicit human computer interaction. The focus is on automatically monitoring activities in the meeting room using audio and video analysis. One of the key SMaRT tools already implemented is the Meeting Browser tool [Bet00], which is capable of automated meeting capture and replay, supporting live meetings in addition to record and replay capabilities. A screenshot of the Meeting Browser is shown in figure 2.10.



```
Col. M. S. Fisher:  THE IRANIAN DELEGATION ASKED TO
                    BE SEATED WITH THE OMAN'S VICE THE SAUDIS AS THE
                    SEATING CHART HAD INDICATED
SecDef John Stone:  WHAT DO YOU THINK THIS MEANS
                    COLONEL
Col. M. S. Fisher:  I'M NOT SURE SIR
Col. M. S. Fisher:  BUT IT JUST COULD BE THAT THE
                    IRANIANS ARE POSTURING TO MAKE A BETTER DEAL FOR
                    THEMSELVES
Col. M. S. Fisher:  WITH REGARD TO THE FLOW OF OIL
                    THROUGH THE STRAITS OF HORMUZ
Adm. Roc Kelly:  IF I COULD INTERRUPT SIR SCOTT WHEN
                 I WAS COMMANDER OF THE MIDDLE EAST FORCE BACK IN
THE
                 MID EIGHTIES THE IRANIANS WERE ALWAYS SPOUTING
                 RHETORIC HOWEVER THIS SEEMS A LITTLE MORE UNUSUAL
Adm. Roc Kelly:  KEEP US POSTED ON ANYTHING THAT
                 COMES OUT OF THAT MEETING
Col. M. S. Fisher:  WILL DO SIR FISHER OUT
SecDef John Stone:  MISS LING I BELIEVE THAT YOU
                    WERE ABOUT TO START
Susan Ling:  YES SIR
Susan Ling:  I WAS GIVING YOU A RECAP ON THE CHINESE
```

Figure 2.10, The SMaRT Meeting Browser (from [Bet00])

The Meeting Browser supports up to six participants, some of which may join remotely using videoconferencing. During a live meeting, the browser displays the participant list and an automatically generated speaker identification timeline and transcript of the meeting. If a list of action points are discussed at the end of the meeting, the system is able to automatically capture these and email them to the meeting participants. The system is also able to automatically generate a text summary from the transcript.

After a meeting, all this information may be archived alongside the video from the meeting, and may be replayed in the Meeting Browser. The collection of archived meetings may be searched by topic, keywords, participants or date, and it is also possible to view the summary for a meeting without having to first load it into the browser.

The functionality of the Meeting Browser is achieved by a combination of techniques. Identification of participants uses computer vision techniques combined with voice recognition. The voice recognition system is also used to generate the speaker identification data. The automatic transcription is achieved using speech recognition software, whose output is combined with the speaker identification data to attribute each comment to the correct participant. The transcript summarisation is achieved through a summarisation server, which analyses the dialogue and returns a summary to the Meeting Browser. Since the voice recognition system is somewhat error prone, with typical word error rates in excess of 25%, the browser allows manual correction of the transcripts.

Type of Collaboration Supported

Support is offered for meetings of up to six participants, who can either be co-located or joined remotely via videoconferencing. The system also allows searching, browsing and replay of archived meetings.

Type of Information Added

The system adds an automatically generated transcript, summary, participant list and speaker event time line. This information is generated in near real-time, and is made available to participants in live sessions as well as recorded sessions. The system is also able to automatically capture discussed action points at the end of a meeting.

Support for Machine Processable Annotations and Semantics

It has an explicit representation of participants speaking and also captures other basic metadata about the meeting such as participants and keywords, which may be used to search for archived meetings. The system also automatically extracts a text transcript and summary from the audio and has an explicit representation for action items.

Support for Live Processing

The system has good support for live processing. The participant list and transcript are both created on-the-fly and displayed to participants during a meeting, although some lag is introduced because the speaker identification requires approximately 6 seconds of speech to produce accurate results.

Degree of Automation

The system is largely automated, supporting automatic participant and speaker identification and automated transcription. However due to the high error rate in the speech recognition, significant human input is required to correct the errors in the transcript.

2.2.6.1 Conclusions for SMaRT

The Meeting Browser has many useful features and supports a significant amount of additional meeting information compared to traditional video recordings. Most of its features are made available to participants during live meetings, which means that its use goes beyond a simple replay tool.

One of its main weaknesses is that its speech recognition has a significant error rate, which requires significant manual input to correct. Its other weaknesses are its limitation to six participants, which may often be exceeded in real meetings and a speaker identification time in the order of several seconds, which introduces some lag in the system.

2.2.7 CoAKTinG

The CoAKTinG (Collaborative Advanced Knowledge Technologies in the Grid) project [Bac04] looks at providing mediated spaces for synchronous collaboration, as well as tools for supporting asynchronous collaboration. The project looks primarily to address the needs of e-Science collaboration, but the work is also applicable to collaboration in a more generic context too.

CoAKTinG applies advanced knowledge technologies to integrate a number of tools into existing collaborative environments, such as the Access Grid [Acc04]. The tools are:

- **BuddySpace.** This is an instant messaging tool with enhanced support for presence awareness. The presence features take the form of a map on to which the presence information is overlaid, allowing users to tell at a glance who is available to chat, and where they are located (shown in figure 2.11). The instant messaging features may be used for 'back channel' communication in videoconferences, and also can support meeting control tasks such as speaker queuing and voting on issues.

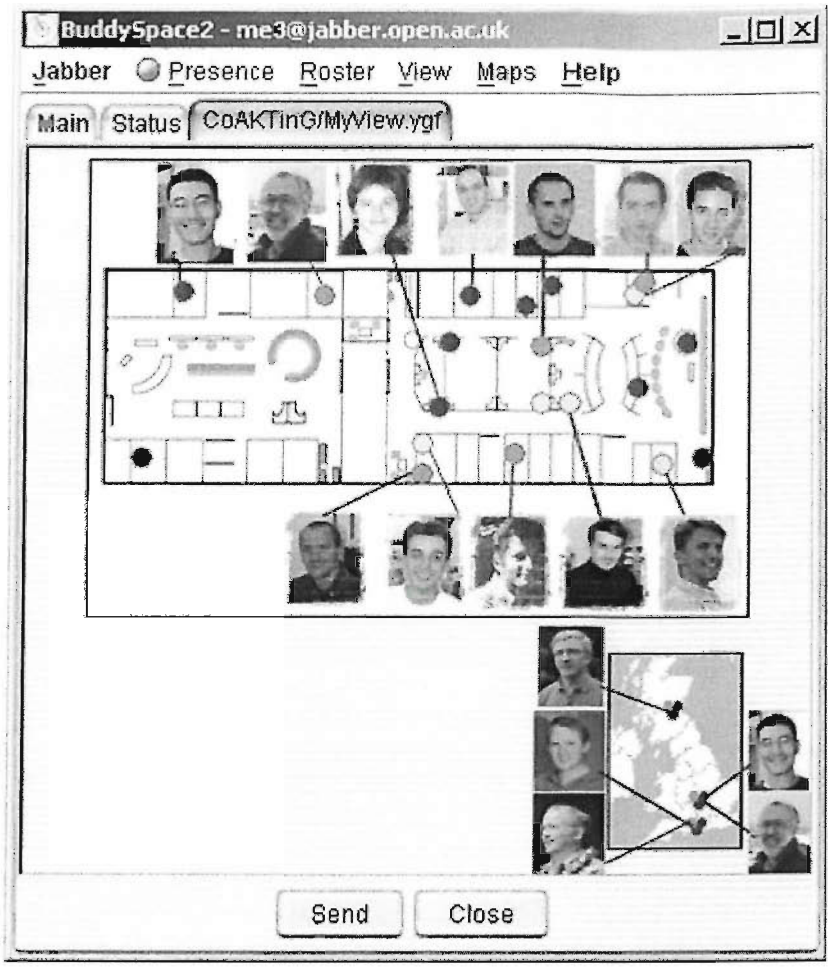


Figure 2.11, The presence indicators of BuddySpace (from [Bac04])

- **Compendium**. This is a graphical tool for collective sense making and group memory capture. Dialogue maps are hand created on-the-fly in meetings by a trained facilitator. The maps consist of interconnected nodes that provide a visual trace of issues, ideas, arguments and decisions in the meeting, which may be validated by participants at the meeting (see figure 2.12). After the meeting the maps provide a structured, searchable group memory for the meeting. Compendium also supports live sharing of dialogue maps to support distributed collaboration and also allows maps to be exported as RDF compliant to an OWL ontology developed for the CoAKTinG project.

- **I-X Process Panels.** In essence, this tool acts as an intelligent 'to do' list, which can be used to coordinate pre or post-meeting actions. The interface shows users a list of their issues and activities, and supports collaboration by allowing the issues and activities to be passed to the panels of other users. Actions may also be created in a meeting specific panel, which are then passed on to the relevant users for action. Upon completion of an activity, users may use their panel to report this back to the meeting specific panel. At its heart, I-X Process panels is built using an activity ontology and has an automatic RDF export function.

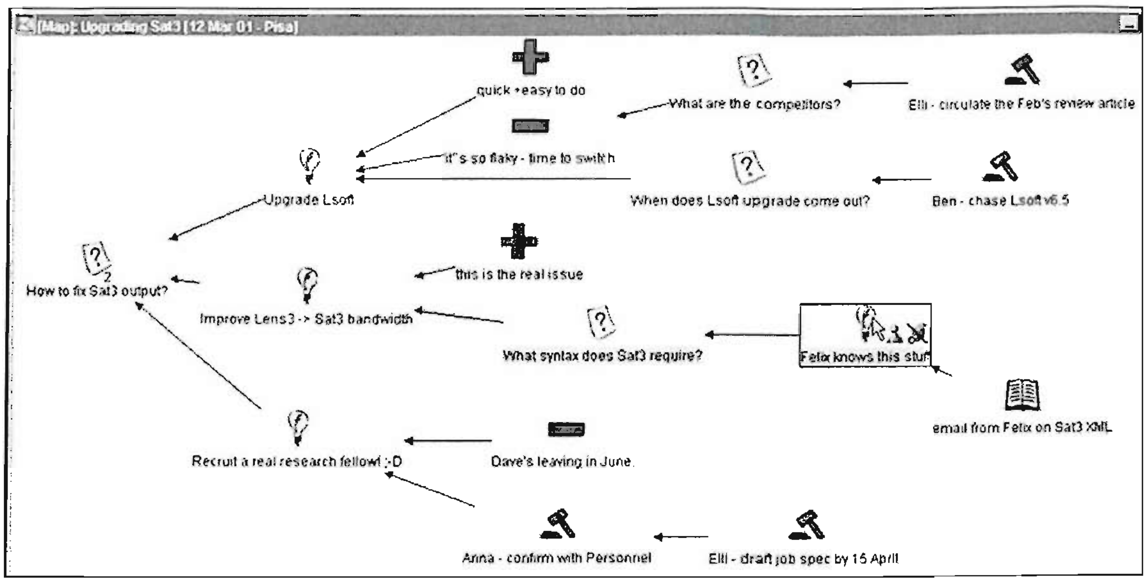
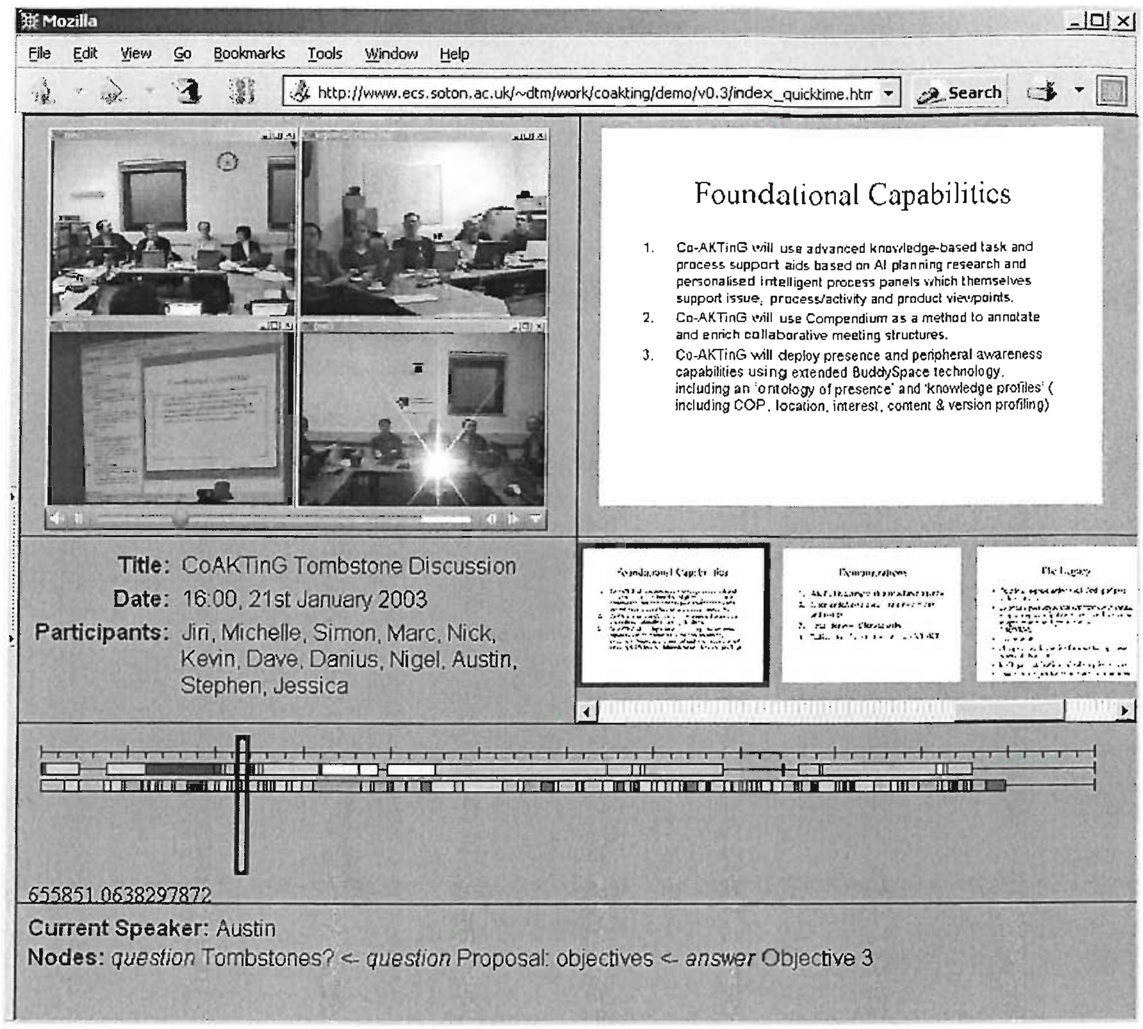

[Map]: Upgrading Sat3 [12 Mar 01 - Pisa]
quick + easy to do
What are the competitors?
Elli - circulate the Feb's review article
it's so flaky - time to switch
Upgrade Lsoft
When does Lsoft upgrade come out?
Ben - chase Lsoft v6.5
this is the real issue
How to fix Sat3 output?
Improve Lens3 -> Sat3 bandwidth
What syntax does Sat3 require?
Felix knows this stuff
email from Felix on Sat3 XML
Recruit a real research fellow! ;-D
Dave's leaving in June.
Anna - confirm with Personnel
Elli - draft job spec by 15 April

Figure 2.12, An example Compendium map (from [Bac 04])

- **Meeting Replay.** Replay of meetings is achieved with a web-based tool that can be used to navigate through an archived meeting (see figure 2.13). The tool takes the recorded video from a meeting along with an RDF description of the meeting events and automatically generates a timeline showing these events. Clicking on an event in the timeline begins the replay of the meeting from that point. The replay consists of the meeting events synchronised with audio and video. The replay tool also displays other useful meeting metadata such as title, date and a list of participants. Events that the replay tool is capable of handling include agenda items, speaker identification, slide transitions and creation of compendium nodes. The RDF description of the meeting events (e.g. speaker identification) is largely created by hand, although Compendium supports automatic RDF export of its maps (which are initially hand generated).


Mozilla
File Edit View Go Bookmarks Tools Window Help
http://www.ecs.soton.ac.uk/~dtm/work/coakting/demo/v0.3/index_quicktime.htm
Search

Foundational Capabilities

1. Co-AKTinG will use advanced knowledge-based task and process support aids based on AI planning research and personalised intelligent process panels which themselves support issue, process/activity and product viewpoints.
2. Co-AKTinG will use Compendium as a method to annotate and enrich collaborative meeting structures.
3. Co-AKTinG will deploy presence and peripheral awareness capabilities using extended BuddySpace technology, including an 'ontology of presence' and 'knowledge profiles' (including COP, location, interest, content & version profiling)

Title: CoAKTinG Tombstone Discussion
Date: 16:00, 21st January 2003
Participants: Jiri, Michelle, Simon, Marc, Nick, Kevin, Dave, Danius, Nigel, Austin, Stephen, Jessica

655851.0638297872

Current Speaker: Austin
Nodes: *question* Tombstones? <- *question* Proposal: objectives <- *answer* Objective 3

Figure 2.13, The web based Meeting Replay interface (from [Bac 04])

Type of Collaboration Supported

CoAKTinG supports both synchronous meeting or videoconferencing scenarios, and asynchronous collaboration after a meeting using I-X Process Panels and the meeting replay tool.

Type of Information Added

Quite a lot of different types of additional information and features are provided. The main features are presence management and visualisation, back channel communication in meetings, shared Compendium maps, issue and activity tracking, and meeting replay. The meeting replay is able to handle event types such as speaker identification, slide transitions and creation of Compendium nodes.

Support for Machine Processable Annotations and Semantics

Being built around an OWL ontology, CoAKTinG has very good support for high level semantics. Both I-X Process Panels and Compendium are capable of generating RDF output which preserves the rich semantic relationships these tools may be used to express. The meeting representations used by the Meeting Replay tool are also created in RDF.

Support for Live Processing

Both BuddySpace and Compendium are tools that can be used in live meetings, and I-X Process Panels allows real-time transfer of issues and activities between panels. The Meeting Replay tool however relies on hand creation of the RDF meeting representation after the meeting has ended.

Degree of Automation

All the CoAKTinG tools rely on a significant amount of explicit user input. For example, Compendium requires significant input from a trained user. The meeting replay tool also requires information such as participant lists and speaker identification to be hand generated after the meeting.

2.2.7.1 Conclusions for CoAKTinG

The CoAKTinG project has demonstrated the use of Semantic Web technologies within the domain of synchronous collaboration. It primarily uses RDF for direct ontology level interoperability of components and expressing relationships to external resources, but doesn't use techniques such as inferencing to realise the full value added potential of RDF.

A significant amount of manual effort is required to use the tools. In particular Compendium and the generation the RDF meeting descriptions for the Meeting Replay tool both involve significant effort. Furthermore, although BuddySpace, I-X Process Panels and Compendium work in live meetings, the reliance of the Meeting Replay tool on hand authored RDF after meeting means that the events it handles cannot be displayed to participants during a live meeting.

2.2.8 Conclusions for Review of Systems

The review has examined a range of systems that provide some form of computer enhanced support for annotation and capture of collaboration activities. Within the context of this thesis, the greatest collective shortcoming of the reviewed systems was an almost complete lack of machine processable semantics associated with the annotations. This severely limits the potential for interoperability, reuse or inference with the annotations. The notable exception to this was CoAKTinG, which had good support for high-level semantics, but did have the drawback that the annotation tools did not all work in real-time and required significant manual input. A lack of automation was a common failing amongst the other systems too, with only Distributed Meetings and AVIARY providing approaches that didn't require significant user input during or after a collaboration session.

NoteLook and Distributed Meetings made very little functionality available during live meetings, and some useful CoAKTinG annotation features, such as speaker identification, were only made available after a session too. This means that session participants would not have access to all helpful annotations during a live meeting. With the exception of SMaRT and CoAKTinG, the systems also did not support full distributed collaboration, which clearly further limits their use.

Overall, the key observation is that none of the systems fully provided all desirable properties at once (i.e. machine processable semantics, live processing and significant automation). Therefore there is considerable scope to create improved systems compared to the ones reviewed here.

2.3 The Semantic Web

The Semantic Web is defined as "an extension of the current web, in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [Ber01]. In essence, this means putting machine understandable data on the web, to enable it to be shared and processed by automated tools as well as people. This potentially enables significantly better automation, integration and reuse of data across a variety of applications.

The key technology behind the Semantic Web that allows the creation of these machine understandable descriptions is the Resource Description Framework (RDF) [Bec04]. It is based on a hierarchical class and property system, where all entities described by RDF expressions are *resources*, which are uniquely identified by a URI (Uniform Resource Identifier) [Ber98]. Resources have *properties* that are specific characteristics or attributes that are used to describe them. The value of a property may either be a literal value or another resource. A resource combined with a named property and its value is a *statement*. Statements in RDF are structured triples of the form (*subject*, *predicate*, *object*). RDF is expressed using an XML-based serialisation syntax, although RDF may also be serialised using other representations such as Notation3 [Ber04].

The RDF *schema* mechanism (RDFS) [Bri04] is used to define the classes of resource that may exist and the properties they are permitted to have. RDFS can be thought of as a mechanism for expressing simple ontologies. In the context of the Semantic Web, an ontology is a representation of a vocabulary, that specifies the terms, their meanings and their interrelationships. Ontologies are typically used for modelling real world domains and therefore function as domain conceptualisations.

The RDF schema mechanism is fairly limited, and to address this, the W3C have developed the Web Ontology Language (OWL) [Dea04]. OWL extends RDFS by enabling the specification of more complex ontologies. It adds additional features for describing properties and classes, such as relations between classes, cardinality of properties, equality, richer typing of properties, characteristics of properties and enumerated classes. OWL also introduces features that allow ontologies to be distributed across many systems, and has standard mechanisms for extending ontologies.

De Roure and Hendler [DeR04] have discussed a number of important aspects of the Semantic Web and these are briefly summarised in here. Much of the added value of the Semantic Web comes from what is known as the Network Effect. This effect comes from the accumulation of available descriptive information about resources. If there are multiple descriptions of specific resources distributed on the Semantic Web, for example held in databases or web sites, then this enables new kinds of questions to be answered that draw on this aggregated knowledge, since it is effectively interlinked by the objects it describes.

They also identify that the current best practice for realising the Semantic Web infrastructure is to use a centralised, persistent and scalable database of triples called a triplestore. This collects knowledge in a single place in a repository that is simple to manage and query. Although this approach works well at present, in the future it is likely that the Semantic Web infrastructure will be provided by many distributed RDF servers that will work with multiple ontologies to remove the requirement of being centrally managed. Rather than having a single monolithic triplestore, the Semantic Web will become a vast distributed triplestore, which will self-organise just as the Web does today, although it is not clear yet how this will be achieved.

2.4 Semantic Web Applications

This section gives an overview of a number of applications that use the Semantic Web. The discussion starts with CS AKTiveSpace, as it is one of the major applications in the Semantic Web field and presents a useful reference for the architecture of the current Semantic Web. It consists of a diverse set of individual services, and the key relevant ones are discussed in the following sections, along with their rationale for selection.

Then the Annotea semantic annotation system and work by the RDF Calendar Taskforce are discussed. Both of these are examples of how the Semantic Web is currently applied to the domain of collaboration, with Annotea supporting collaborative annotation and the calendar work supporting automated scheduling of collaboration sessions.

2.4.1 CS AKTiveSpace

Arguably one of the more important applications that goes some considerable way to demonstrating what the Semantic Web can offer is CS AKTiveSpace [sch04]. This is a large scale proof of concept application to demonstrate what the Semantic Web can provide and is built on a larger scale than existing implementations of individual Semantic Web services.

The application exploits a wide range of semantically heterogeneous and distributed content relating to Computer Science research in the UK. It uses a single common ontology called the AKT reference ontology [AKT04] to integrate the different data sources. The content is gathered on a continuous basis using a variety of methods including harvesting from existing databases, scraping from institutional websites and direct submission. Specific mediators for each data source are used to convert the information obtained from the sources to be in terms of the ontology.

CS AKTiveSpace attempts to address a number of key Semantic Web issues including harvesting, time performance of queries, robustness, scalability and referential integrity. Referential integrity issues arise when more than one URI is used to represent a single resource and is a particular problem in applications like this, as knowledge is integrated from multiple sources. While such co-references are entirely permissible in the Semantic Web, they are problematic as they partition the information space in a way to reduce the recall of queries made to that space. For existing information sources in CS AKTiveSpace, a combination of manual and automated heuristic techniques are used to identify co-references and rectify them, but in the future plans for CS AKTiveSpace hope that the knowledge base will be used as a gazetteer or naming authority to ensure that agreed names are used for resources.

2.4.1.1 Choice of Services for Discussion

CS AKTiveSpace consists of a large number of services. An overview of the chosen services is given here along with the rationale for their choice.

- **3store**. This is a triplestore implementation that provides the core knowledge repository for CS AKTiveSpace. It was chosen as it is at the heart of CS AKTiveSpace and any discussion would be incomplete without it. Furthermore it is used as part of the implementation discussed in Chapter 5.

- **Armadillo**. This service was chosen as it shows one current approach to semi automated semantic annotation of Web Resources.

- **Ontocopi**. This service is used to automatically determine the Communities of Practice of individuals. This has particular relevance to collaboration as it can be used to identify new people to collaborate with.

2.4.1.2 3store

Content in CS AKTiveSpace is held in a centralised triplestore called 3store [Har04], which at the time of writing contains about 10 million triples. 3store has been designed with scalability and performance in mind and it can scale to the order of 25million triples and answer typical queries in a few milliseconds. Queries are issued using the commonly used RDF Data Query Language (RDQL) [RDQ03]. 3store also has a built in inference capability, so that when it is queried, it not only returns the triples explicitly asserted in the triplestore, but also any triples that may be entailed from the RDF and RDFS language rules, which depending on the nature of the particular entailment are either worked out at assertion time or dynamically at query time.

2.4.1.3 Armadillo

One of the services used to constantly update the CS AKTiveSpace triplestore is called Armadillo [Cir04]. It is an application for largely automated knowledge extraction from web pages. It retrieves information from different sources and integrates it into its repository. The repository can be used both to access the extracted information and to semantically annotate the web pages where the information was identified.

It has an initial lexicon for recognising instances of concepts, and it then can automatically expand its lexicon by exploiting patterns in the data set it is processing. It also exploits redundancy of information on the web to expand its lexicon and improve the accuracy of its information extraction. The only user input required is to add information missed by the system and to delete information incorrectly identified by the system. This user intervention feeds back into the system to improve its future effectiveness.

In CS AKTiveSpace Armadillo is used for extracting the names of researchers and paper citations from institutional web sites.

2.4.1.4 Ontocopi

The CS AKTiveSpace infrastructure is used by a number of applications. An especially useful one is the Ontology Based Community of Practice Identifier (Ontocopi) [Ala03]. It is an application that demonstrates the value of the network effect by automatically identifying communities of practice (COPs), that otherwise would be extremely difficult to determine. Communities of practice are self-organising informal groups of individuals interested in a particular job, practice or work domain. Knowing COPs is often important within organisations, as they help with understanding the knowledge resources of an organisation, but determining them can be difficult and time consuming.

Ontocopi uses ontological relations to infer connections between objects that are only implicitly represented. E.g. that two people work with the same people, go to the same conferences or have published in the same journal. These relations are determined using a technique called Ontology-based Network Analysis (ONA), which determines sets of instances associated with a specific instance in a knowledge base. It obtains the COP of a selected instance by traversing selected semantic relationships between the instance and other instances, continuing recursively until the links are exhausted or a link threshold has been reached. The algorithm is general purpose, so it is not only possible to determine the COP of people, but of any instance in the triplestore, such as a project.

Another interesting use for COPs is for resolving referential integrity issues. When the COP of two instances is sufficiently similar then it proves that the two instances are identical.

2.4.2 Friend Of A Friend (FOAF)

Friend Of A Friend (FOAF) [Bri05] is a Semantic Web vocabulary for specifying social networks. It allows individuals to create machine-readable homepages that describe people, the connections between them and the things they create and do. This allows software tools to automatically aggregate this information and harness the network effect to infer relationships between people and resources linked to those people, even though those relationships may not be explicitly specified anywhere.

For example, FOAF could be used to automatically sort a person's emails by prioritising the messages have been sent from individuals who are have an some form of link (either explicit or implicit) to that person. Other potential applications could enable people to automatically identify individuals with the same interests as them, or perhaps automatically determine the complete set of authors for a document, or the set of people who are co-depicted in the same photograph (even though none of this information will have been explicitly specified in any single location).

Tools such as FOAF-a-Matic [FOA05a] can be used to assist individuals with the creation of FOAF content. Services such as FOAF explorer [FOA05b] can be used to view and navigate the network of FOAF information.

Arguably one of the biggest weaknesses of the current FOAF specification is that its mechanism for describing explicit links between people has very limited semantics, being restricted to just a single type of 'knows' relationship. This makes it difficult to determine the differences in relationship types between people. This, for example, would make it impossible to differentiate between knowing trusted work colleagues and knowing casual acquaintances, when each category of relationship should ideally be treated differently in the social network.

2.4.3 Annotea

Annotea [Kah01] is a system from the W3C for the asynchronous collaborative semantic annotation of Web documents. Users may annotate specific sections of a document and these annotations are then made available to other users viewing the document, who may author further annotations. The system uses an RDF based infrastructure, where the annotations are held in annotation servers, which are just general purpose triplestores accessible via HTTP. XPointer [Gro03] is used to specify which part of the document has been annotated. Annotea specifies a core RDF schema that defines a number of different annotation types, such as comments, questions and advice. Users can use the standard RDF extensibility mechanisms to add other annotation types that are required for their individual needs or the needs of their community. In addition to annotations, Annotea also supports shared bookmarks [Koi03], to provide a collaboratively maintained list of links to interesting Web documents displayed in a hierarchical category view. The bookmarks may also be displayed in context within a document to provide links to related information about a concept within the document. As with annotations, the bookmarks are stored in general purpose triplestores.

Annotea only specifies the infrastructure, and it is left to the individual client implementations to determine how the functionality should be presented to the user. Annotea capable clients are typically Web browsers, which also allow the authoring of annotations and bookmarks. When a client fetches a web page, it also queries one or more annotation servers to retrieve the annotations for the page. In order to achieve this, the client needs to be pre-configured with the locations of the of the annotation servers. Annotea clients include Amaya [Ama04] and the Annozilla plug-in for Mozilla [Ann04].

One shortcoming of Annotea is that the author of an annotation is stored as a literal name rather than a URI representing that person. Clearly this makes it very difficult for information about the author to be reliably retrieved from the Semantic Web, and thus is unable to fully harness the network effect.

2.4.3.1 Vannotea

Vannotea [Sch03] is a system based on Annotea for the real-time, synchronous collaborative annotation of high quality video streams. It supports multiple distributed users who can communicate using the Access Grid, although the system is independent of the Access Grid, so any real-time communication technology could be used in its place. The system is presented to each user as a video player window where they may collectively watch and control the video in question (see figure 2.14). An annotation and discussion window allows users to author textual annotations, which may refer to a segment of video, an individual frame, or a region within the frame. This window also displays any existing annotations for the current video segment.

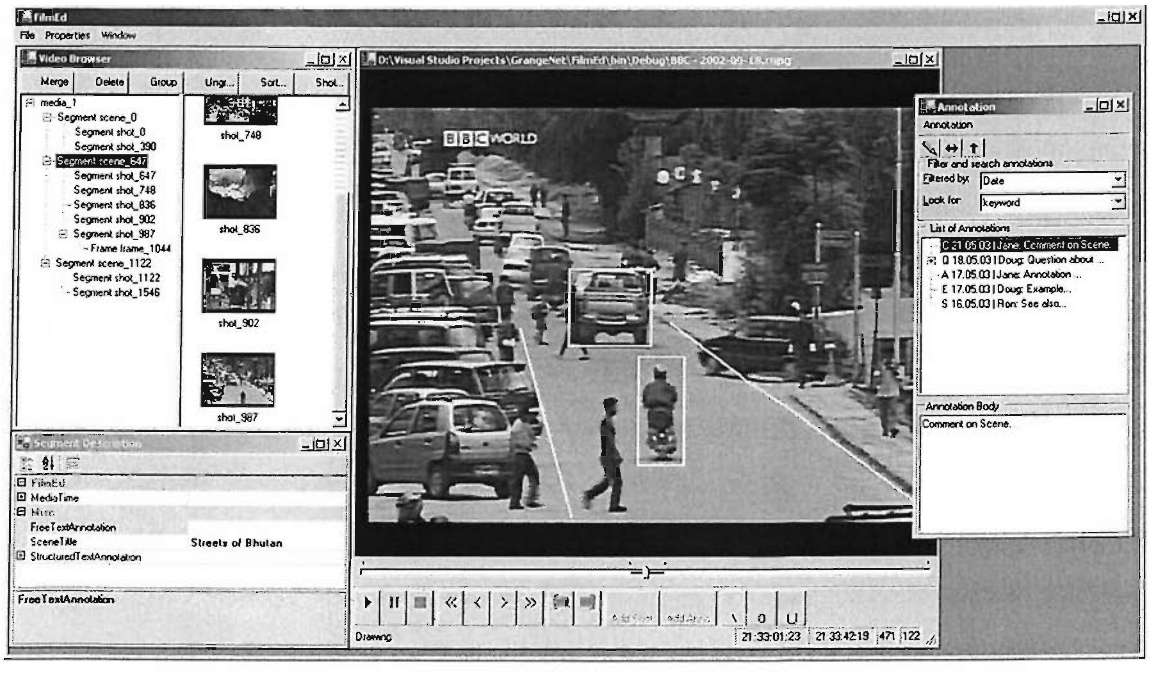


Figure 2.14, The Vannotea video annotation interface (from [Sch03])

The textual annotations use the Annotea system and are stored as RDF in an annotation server, and Vannotea uses an extended form of XPointer to refer to specific segments, frames or regions of video.

Vannotea's integration into the semantic web is in fact fairly minimal. It treats Annotea like a 'black box' annotation service and doesn't take advantage of its semantic features. Furthermore, although the individual annotations are in RDF, the metadata used to describe individual video files (e.g. for locating the files in the first place) is in a plain XML format, rather than RDF.

2.4.4 RDF Calendar Taskforce

The RDF Calendar Taskforce [Pay02a] has worked on creating ontologies and tools to support calendars on the Semantic Web. The purpose of this is to allow software agents to automatically understand and reason about calendar events and schedules, which has many advantages such as being able to find mutually agreeable appointments for several attendees, determining where events occur and who is attending. The network effect can then also be used to tap into other knowledge on the Semantic Web such as the attendees connections and affiliations. The ontology work has focused on creating a calendar ontology based on the widely used iCalendar format (RFC 2446). One of the taskforce's key tools is RCAL [Pay02b], which uses the calendar ontology to allow browsing, importing, automatic scheduling between multiple users and collation of knowledge obtained from multiple sources.

2.5 Conclusions from Literature Review

The concept of a mediated space has been introduced and it has been shown that mediated interactions could potentially be as effective as, or possibly more effective than non-mediated interactions.

Six existing systems that supported annotation and capture of collaboration activities have been reviewed and it has been shown that all, except the CoAKTinG tools, lacked machine processable semantics. Furthermore, a significant number of the systems had poor support for distributed collaboration, live processing and automatic generation of annotations. This shows that there is considerable scope for improving these applications.

The Semantic Web has also been covered and it has been shown that it is still a relatively new research area and that the exact form it will take once it has matured is still unclear. CS AKTiveSpace has been discussed as an example of a large scale proof of concept Semantic Web application. One particular Semantic Web issue that is unclear is that of triplestore discovery, and existing applications such as CS AKTiveSpace and Annotea rely on triplestore locations being manually specified.

The Semantic Web is being used for different forms of collaboration, such as document annotation, identification of communities of practice and scheduling of meetings. However, on the whole, Semantic Web technologies have not been applied to temporal media and have not been used for real-time synchronous collaboration. For example, although CoAKTinG tools and Vannotea both support synchronous collaboration, CoAKTinG primarily uses Semantic Web technologies after a meeting for the purposes of archiving. Vannotea's integration into the Semantic Web is in fact only minimal, as it treats Annotea like a 'black box' annotation service, and other metadata used by the system does not use Semantic Web standards.

3 Background and Motivation

This chapter explains and motivates the use of semantic annotation for enhancing distributed real-time collaboration. It describes what annotations can be used for in this context and what the potential benefits of a semantic approach are. To further motivate real-time annotation of collaboration sessions, a small scale study of text based IRC chat usage in telephone conferences is presented. This study is used along with the author's own experience to produce an example list of useful annotations. Finally an example scenario is presented in which it is shown how semantic annotation can be used to enhance Access Grid videoconferencing, and potentially other remote conferencing technologies too.

3.1 Introduction

The work in this thesis builds on the concept of *continuous metadata* [Pag01]. That is, temporally significant metadata that is transported in close synchronisation with streamed multimedia data to be used as supporting information to enrich the multimedia data. Continuous metadata has been demonstrated by the HyStream application [Cru01], which used hypertext links as an example form of metadata. The application was capable of delivering the links synchronously with multimedia streams over a wide area network. A demonstrator was produced that was capable of synchronising links to presentation slides with recorded seminar videos. Later extensions to HyStream [Bea01] enabled it to use a simple RDF schema and interact with an RDF knowledgebase. This allowed automatic generation of a user interface for hand authoring the temporal links, which reduced authoring effort

This existing work focused on the offline, hand mark-up of recorded media. The RDF based extensions were also very basic, for example not incorporating any notion of time in the schema itself.

3.2 Semantic Annotations

In the context of real-time distributed collaboration, semantic annotation means giving the individual events that occur as part of a real-time collaboration activity an explicit representation that has a formally defined meaning. Annotations are generated during collaboration activities and, if appropriate, can then be displayed to session participants in real-time. For example, in addition to distributing audio and video streams between sites in a videoconference, a real-time generated description of the events in the session is also distributed between sites, and these events are presented to session participants in a suitable format.

Such semantic annotations have two key purposes, firstly to provide useful additional information in real-time for session participants and secondly to provide a machine understandable description for a session, which can be used to index recordings of collaboration sessions and then be replayed in synchronisation with the audio and video recordings to provide a more complete replay that audio and video alone could provide.

3.3 Supported Technologies

This work aims to be as independent as possible from any particular collaboration technology. The main type of technologies it aims to support are those for multipoint, group-to-group, real-time collaboration. In particular this includes videoconferencing (e.g. Access Grid) and audio conferencing (e.g. telephone audio conferences). The primary focus is on videoconferencing, as this is a method of collaboration that is becoming increasingly popular and is a field in which the author has much first hand experience.

3.4 Supported Collaboration Types

Here the main area of support is for synchronous (i.e. real-time) distributed collaboration, but also asynchronous collaboration through allowing semantic annotations to be used for archiving and later replay of collaboration sessions.

There are a broad variety of activities that fall under the category of distributed real-time collaborations. For example, surgeons collaborating during a live operation will have requirements very different from those of computer science researchers discussing an academic paper. There are also different modes of collaborating in real-time, e.g. informal group discussions, seminars with a single presenter and an audience, or more formal meetings.

The intention is to be general purpose enough to support a wide spectrum of different collaboration activities. From the author's own experience, a common use of real-time distributed collaboration technology, at least in the academic field, is for group discussions, which are reasonably informal, though may still have a chair. It is such general-purpose group discussions that will form a focus for the work covered in this thesis.

In terms of scale of collaboration, it has been chosen to use the author's first hand experiences of Access Grid collaboration to provide sensible figures for the size of collaboration sessions this work should aim to support. A typical Access Grid session may have approximately 10 participants and consist of 3 or 4 sites, with an upper limit of about 12 sites and 30 participants. These figures will be used as a basis for the scale of collaboration that this work should be able to support, although the work aims to be general-purpose enough to support smaller or larger scale collaboration.

This work is also applicable to some extent to situations where participants are all co-located in the same physical space. Distributed collaboration has been chosen as the focus of this work as it is often less effective than face-to-face collaboration, meaning that there is a greater need and more potential for improvement. Furthermore, since the collaboration is already being mediated by technology, it makes sense to try and improve how that technology performs the mediation.

3.5 Motivation For Annotation

From a human perspective, the main reasons for annotation are to provide useful additional information to session participants and to provide an index and archive of a recorded session. Annotations could, for example, be used to provide information about the following:

- The current agenda item.
- Information about the current set of participants, such as a list of names.
- When somebody is explicitly addressing you.
- When the group is bored. This could be useful for somebody who is presenting so they can gauge when to move on to the next topic.
- When participants are distracted. There is no point addressing a remark to somebody if they are not paying attention to hear that remark.

- When somebody is lying (e.g. obtained from polygraph or voice stress measurement). This could have applications in legal settings.

The net result of adding live annotations should be an improvement in the level of conversational and workspace awareness among participants (see section 2.1.4). Annotations can be used to explicitly provide information that is lost in video or audio conferencing because of missing perceptual cues. Important missing cues are factors such as audio direction and gaze direction, which can make it difficult to tell who is speaking or who they are speaking to, therefore reducing conversational awareness. Cues like these have been described as *focal assurance cues* [Man97] and give information relating to each participant such as who is speaking, asking questions or interrupting. "In situations where the participants are not familiar with each other it is especially hard to develop a sense of where people stand on issues when contributions are not tied to a specific participant" [Man97]. Annotations, such as explicit speaker identification could be used to compensate for these lost cues.

Basic workspace awareness, such as knowing exactly who is in the session and what they are doing can also be difficult to maintain, as not all participants may be on camera. Annotations provide a mechanism to enable participants to obtain information at a glance such as who is currently in the meeting and what is currently happening, thus enabling them to maintain their levels of awareness.

Furthermore, annotations can go beyond just replacing those cues missing in videoconferencing. Through displaying explicit annotations that provide information that is only otherwise implicit, the potential exists to boost participants' levels of awareness to beyond those found even in face-to-face communications (as discussed in the review of Hollan and Stornetta's work in section 2.1.5). For example, in meetings (either face-to-face or video mediated) it may only be implicit that the group is bored, meaning that this might not be noticed by a presenter. However, an explicit annotation notifying a presenter of this fact would perhaps allow the presenter to modify their presentation to try and recapture the interest of the group.

Once a session has finished, the annotations can then be used in tandem with audio and video recordings to serve as an archive for the session. Traditionally meeting archives have consisted of meeting minutes, which serve as a compact, structured record, but one that often leaves out many of the subtleties of the meeting, and there is no way of determining the rationale behind decisions if it has not been recorded. Audio or video recording can overcome this problem, but present large amounts of unstructured data, much of which may be irrelevant to the viewer.

These problems can be addressed by annotations. They firstly can be used to index the audio and video to, for example, begin replay after a certain person joined the meeting or to replay all the sections when a certain person spoke. They could even be used by people who were present at the meeting answer post meeting queries of the type "replay all the sections meeting where I was distracted", thus allowing them to catch up on what they missed. Furthermore, the annotations can be replayed in synchronisation with audio and video to provide a more complete replay, for example showing the current agenda item, a list of all the participants present at that point in the session, or even perhaps showing when somebody was lying.

3.5.1 What is being annotated?

Annotations normally need some entity to be annotated. Here the primary entities being annotated are the actual events that make up the collaboration activity. This annotation of actual events holds true for both face-to-face and video mediated collaboration. If the collaboration is video mediated or is being recorded, then the media streams (i.e. audio and video) will be further entities that are being annotated in addition to the actual meeting events. For example, the event of somebody being distracted is treated here as the entity being annotated. If this event is recorded in video, then the annotation will also serve as an annotation for the video.

When describing annotations in this thesis, the author refrains from referring to them as 'metadata', since this might incorrectly imply that there was always some underlying explicit data being annotated. In a face-to-face meeting that isn't recorded there is no explicit data, so describing the annotations as metadata could cause confusion. It is however true that when a meeting is video mediated there is explicit data (in the form of video), for which the annotations can act as metadata for.

3.6 Motivation For A Semantic Approach

At this stage it may not be clear why it is beneficial to take a semantic approach to the annotations, i.e. one that is formally defined by an ontology giving them a machine understandable meaning. In addition to the potential for the network effect discussed in section 2.3, a semantic approach has a number of key benefits:

- **Inference.** Inference is the process of deriving new knowledge from that which is already known. This means that new events (and hence annotations) may be automatically derived from the events already known to the system. Automated inference is only practical when a formally defined ontology is used. Inference may be a useful technique for the automatic generation of semantic annotations.

- **Interoperability and Reuse.** A semantic approach means that a system can seamlessly integrate with existing Semantic Web knowledge sources, such as triplestores. This knowledge can be automatically harnessed when creating annotations and can feed into the inference process, to fill in knowledge gaps that would otherwise prevent certain useful inferences being made. By reusing existing knowledge from the Semantic Web this reduces the amount of information required to explicitly bootstrap the system and potentially gives the system access to a wider breadth of knowledge than would otherwise be available to it. Furthermore, systems that support distributed collaboration are inherently distributed themselves. By having a common ontology, it ensures that distributed, heterogeneous components are able to communicate.

- **Extensibility**. A point related to interoperability is extensibility. Systems over their lifetime are often extended or modified, and often in the case of distributed systems not all components are upgraded at once. By using technologies from the Semantic Web, standard extensibility mechanisms may be used. This means that for components that are not upgraded, on receipt of a concept it does not have knowledge of, it may fetch the unknown ontology via the web and use techniques such as transitive closure, to navigate back through the class and property hierarchy until it reaches a concept it does have knowledge of. The new concept can then be treated as an instance of the known concept, with the extensions to the concept ignored.

- **Indexing**. An important use for semantic annotation is to provide a machine understandable description of a collaboration session. This description has uses both in live collaboration sessions and for archived sessions. In live sessions it could, for example, be used as a pattern to match certain sections of archived material, which may be useful to display in the live collaboration. For archived sessions it can be used as a temporal index, allowing users to locate specific sections of interest by event type or could be used to perform further offline inferencing.

3.7 Motivational Study of W3C Telephone Conferences

To provide motivation for the real-time annotation of live collaboration sessions, a brief study was made of some of the W3C's telephone conferences (telcons). These telcons were of particular interest because they usually used a text-based IRC (Internet Relay Chat) session in tandem the telephone audio. The IRC sessions were used for back channel communication during the telcons, and as such could be thought of as providing a rudimentary form of temporal annotation for the telcon. Furthermore the W3Cs telephone conference bridges support two different IRC bots, which can join the IRC session and be commanded by participants to perform useful meeting functions. The bots provide output as further IRC chat entries, which can also be thought of as further basic temporal annotation of the telcon. The first of these bots is called Zakim [Kot04], and it supports the following useful features:

- **Showing participants joining and leaving the telcon**. This is achieved by using caller ID data, and each telephone number can have a name associated with it. If a person joins who is not yet known to the bot, it can be told who that person is. The bot can even be told that several people are sharing a phone at a given site. The bot can be queried at any point in the conference to find out who is present in the telcon. Another interesting feature of the bot is that it can be queried to find the country that each participant is currently in. It does this by using the dialling code of the telephone numbers.

- **Agenda tracking**. The bot can be told the list of agenda items for the session, either by entering them directly in the IRC or by passing it a URL that points to a file specifying the agenda in RDF according to a simple schema. The bot then can keep track of the current agenda item by being informed when the current agendum changes or when an agendum is closed. The bot also has a future reminder (or 'ping') feature that means it can be told to remind the participants about some issue at a time later in the session.

- **Floor control**. Participants can indicate their desire to speak by joining a virtual queue and the chairperson then selects people from the queue to speak. The bot can also be configured to limit the amount of time each participant may speak for.

- **Control of the telephone conference**. The bot can be told to mute or disconnect telcon participants and can also be queried for the telecon pass code. It also has a feature that can determine the current active audio sources in the teleconference. This feature is primarily used to determine sources of feedback and noise in a telcon, but can also be used to find out who is talking, which could be useful for participants who do not know all the other participants.

- **Scribe nomination**. The bot can be asked to randomly select one of the participants to act as scribe for the current meeting.

The Zakim bot is usually used in conjunction with a second bot called RRSAgent [Swi04], which automatically creates a web accessible persistent log of the IRC session. The bot records the session as plain text, HTML and RDF. The RDF schema is very basic and only records IRC chat events (both human and bot generated). Each chat event consists of a timestamp, the text from the IRC entry and the IRC nickname of the person (or bot) that created the entry. RRSagent has the additional feature that it can track action items while a meeting is in progress. This is achieved by a participant entering the action item to the IRC and prefixing it with the text "ACTION:".

In order to see how the IRC and bots were being used in real telecons, the IRC logs of ten telcons were examined to see which bot features were used most frequently and to see what kinds of information were exchanged in the IRC channel during the telecons. It is likely that the features that were used most frequently were also the most useful. A table showing the usage data extracted from the IRC logs is given in Appendix A and a transcript from one of the IRC sessions is given in Appendix B.

3.7.1 General Observations

The observed telcons had fairly high numbers of participants ranging from 9 to 29 people, with an average of 17. Entries (both human and bot created) appeared in IRC on average once every 18 seconds, and given that the telcons typically lasted 1.5 hours, this fairly heavy usage shows that the IRC was a useful collaboration tool for participants.

3.7.2 IRC Bot Features Used

The most frequently used Zakim feature was the 'who is here' function. This is not surprising as the telcons examined were all quite large, so that keeping track of who was present could be very difficult, hence this feature seems to have been extremely useful. Another frequently used feature was the ability to manually specify the names of the people dialling who were not already known to the Zakim bot. It appears that the value added by being able to tell by name who was in the session justified the additional effort of manually entering this information.

Another feature that was used in every conference observed was the speaker queue. The frequency with which it was used clearly shows that participants must find it of use. Given the large numbers of people in the telcons and the absence of any visual information, it is unsurprising that this feature was so popular, as without it there could potentially be many people all trying to speak at once. Conversely, the speaker time limit function was not used once in the logs examined. This probably reflects the reasonably informal format of the meetings, where speaking to a fixed time was not crucial. A surprisingly popular feature was the random scribe nomination feature, which was used in over half the conferences (the author expected that more 'scientific' means might be employed).

Zakim's Agenda tracking features were used in over half of the sessions. The reason they were not used in more sessions may have been because the mechanisms for initially specifying the agenda items were not very user friendly. The future reminder 'ping' function was not used in any sessions.

The ability to identify audio sources was also used in half the sessions. This was due to a high incidence of audio problems in the teleconference (e.g. feedback, noise etc.), which seems to be a problem for such large scale conferences. It appears that such a feature is useful, at least for large teleconferences, which are more prone to technical problems.

The ability to geographically locate a dialling code was not used. The reason for this is probably because the participants in the working groups already knew each other and also that other than satisfying somebody's curiosity, geographically locating a participant wouldn't be of particular use in a session.

Muting via the Zakim bot does not seem to have been used very much either. This is probably because the feature was also available through telephone key presses, which may have been perceived as a simpler way of controlling the function.

The action item specification feature of RRSAgent was only used in three out of the ten sessions examined. This relatively low level of usage indicates that this feature may not have been very useful (often the IRC was used to directly specify action items instead).

3.7.3 Non-bot Related Information Sent in IRC

In addition to the features of the Zakim bot and RRSAgent, the IRC was also heavily used as a back channel for text communication during the telcons.

In the majority of the conferences examined, the IRC was used a mechanism for commenting on the current issue being discussed in the telcon. The advantage of using the IRC for this is that the speaker can continue without being interrupted and may be able to address the issue at an appropriate moment.

The IRC was also heavily used for as a medium for scribing the session. The possible advantages for this could be that participants can see the scribing as it takes place, so can check that they agree with it and have it as a source of textual reinforcement of what is going on in case their attention wanders. Another advantage is that since the IRC sessions are typically archived automatically using RRSagent, it removes the need for the person doing the scribing to have to manually distribute or archive the notes. The IRC was also used as a medium to communicate the status of participants (e.g. to indicate that they will be back in 5 minutes etc).

Another important use of the IRC channel was to distribute URLs during the sessions. Interestingly, the most popular target of such URLs were to emails in the W3C mailing list archives, typically from the same working group that the meeting was for. The URLs were used as pointers to emails from the mailing list that were relevant to the discussion in the telcon. In over half the sessions, a URL was used to distribute the agenda for the telcon. The agenda was originally distributed before the session by sending an email to the mailing list of the working group. Then at the start of the telcon, a URL to the email was posted to the IRC channel by one of the participants as a reminder of the agenda. This use of URLs to archived emails within a telecon was an interesting bridge between the asynchronous collaboration of emails and the synchronous collaboration contained within the telcon. In addition to URLs to archived emails, URLs to documents were also distributed when the documents were relevant to the discussion.

IRC was also commonly used to directly communicate agenda items (bypassing Zakim's agenda tracking features) and to indicate when agenda items had been closed and to indicate action items (bypassing RRSAgents action item features). The IRC was also used to discuss who would be scribe, which often complemented using Zakim's automatic scribe nomination features. For example, if the automatic scribe nomination was used to determine who would be scribe during the next meeting, IRC was used to confirm that the particular person would be present in the next meeting. IRC was also used in three of the meetings to indicate that there were technical problems with the teleconference.

3.7.4 Conclusions

The heavy usage of the IRC and bots (especially Zakim) in this real-world application domain, provides strong evidence that live temporal annotation of collaboration sessions is a useful feature for participants. Furthermore, the usage of RRSAgent to record these sessions, provides evidence that archiving temporal annotations for future reference is useful also.

The main weakness of the IRC and bot approach examined here are the lack of high-level semantics and the requirement for hand-authored annotations. Although RRSAgent was able to export the IRC logs as RDF, the schema used was very basic. For example, information generated by Zakim was treated exactly the same as any other plain text IRC entry, and the authors of the IRC entries are just recorded using their IRC nicknames. Clearly this limits the scope for any further automated processing on the IRC data.

The reliance on hand-authored annotations and hand issued commands to the bots is also far from ideal, as it required significant effort from participants. Additionally, users required some significant prior knowledge to enable them to use the bots, which would prevent users that did not have that have that knowledge from getting the maximum benefit from the bots.

3.8 Examples of Collaboration Events

The events that go to make up an individual collaboration activity are dependent on the nature of the activity taking place. There are however a number of events that will be common to a significant number of different collaboration activity types (in particular group discussion type activities), and some of these events make useful semantic annotations. This section presents some examples of common events, and discusses how they may be useful as semantic annotations. This list has been compiled from the author's own experiences and observations from Access Grid sessions and from W3C telcons. While this list consists of the most obvious events, it is not exhaustive and it may be possible to come up with other useful events in future.

- **Individual people leaving or joining the meeting.** Sometimes due to other commitments, people join or leave meeting mid session. Having this explicitly flagged as an annotation is useful as, since not all participants are always covered by a camera, and it might not always be obvious when somebody has joined or left. It is also useful for indexing archived sessions as it can be used to locate the section of a meeting after a specific person joined, or if a participant had to leave part way through, they can easily watch a replay of the section after they left at a later date. This information can be presented as a dynamically updated participant list, with recently joined participants highlighted. This allows participants to tell at a glance who is in the session, which helps general awareness. Such a list of names is also useful as it can, for example, help if a participant has forgotten another participant's name.

- **A person speaking**. Explicitly identifying who is speaking makes up for lost perceptual cues such as audio direction. If the identification is by name, it can also help put a name to a face, which could be useful when the meeting participants do not know each other in advance. This can also be useful for indexing as it can be used to, for example, locate sections of a meeting where a specific person was the main speaker.

- **The start and end of the meeting.** Annotations that represent the start and the end of the meeting could be used by a media recording component to determine when it should start and then end its recording of the session. The information could also be used by a signage display screen outside the videoconferencing room to show that a meeting is in session and that the participants should not be disturbed.

- **The current agenda item.** This is useful for increasing participant awareness and helping participants who are not paying full attention. This is also useful for indexing as it allows navigation of recorded media by agenda item.

- **A slide being displayed**. When the meeting uses slides as presentation materials, it is useful to share slide transitions to achieve synchronised display of the slides at each site. For archived sessions, slide transitions can be used for synchronised replay of presentation materials with the media streams. They also have additional use for indexing, allowing a user to select a slide and replay the media associated with that slide.

- **A resource being relevant to a specific section in a meeting**. In some meetings, external resources such as documents or images may be relevant to certain sections of the meeting, either because they are being explicitly discussed, or just in more general terms. Annotations containing references (e.g. URLs) to the resources may be distributed to the computing devices of the session participant to allow them to easily view the resources. Similarly, this provides an easy way to access the relevant resources during replay. Furthermore, this could be used for indexing, where the user could be presented with a list of resources associated with the session and may select to replay the section associated with a particular resource.

3.8.1 What constitutes an event?

At first glance it may seem that some of the annotations from the list in the previous section are not events at all. For example, a document being relevant to a specific section in a meeting may not appear to be an event. Despite appearances it is in fact an event. The event is that document being relevant to the meeting, and that event has a start time when the document starts to be relevant and an end time, when it ceases to be relevant. Similarly, each agenda item is an event, which starts when the meeting reaches that agenda item and ends when the meeting moves on to the next agenda item.

So in general terms, an event in this context is something that occurs for a time interval with a defined start and end time. This means that unlike the list in the previous section would suggest, the start and end of the meeting are not treated as individual events, but the whole meeting is treated as a single event that lasts the duration of the meeting. Likewise a person joining or leaving a meeting is treated as a single event, the event is that person being present in the meeting, which will have a start and as end time.

It is also worth pointing out that the current agenda item and the slide being displayed are just special cases of a resource being relevant to a specific section in a meeting, since an agenda item or a slide are both resources.

3.9 Motivational Access Grid-based Scenario

This section describes the addition of a number of different annotation types to Access Grid videoconferencing as a motivational scenario for live semantic annotation. These annotations include displaying the attention levels of individual participants, the group's current level of interest, identifying when the meeting is overrunning, participant tracking and speaker identification. A scenario involving the last two annotation types from this list has also been discussed by the author in [Jub03] and an implementation of participant tracking and speaker identification functionality (but without the window highlighting described in this scenario) is described in chapter 5 of this thesis. The scenario presented here is reasonably generic and a significant portion could be applied to other videoconferencing systems and even audioconferencing.

These areas of common ground between technologies will be discussed in this section too.

3.9.1 Access Grid Background

The Access Grid is a room-based videoconferencing system that enables large-scale group-to-group interactions. Each Access Grid installation is known as an *Access Grid node* and at the time of writing there are over 250 of theses nodes worldwide, with the number growing continually.

The Access Grid runs on standard PCs and uses the Internet's multicast backbone (Mbone) as the transport mechanism for the media streams. Multicast to unicast bridges are provided for sites that do not have multicast connectivity. Audio and video are handled by special versions of the Mbone conferencing tools *rat* and *vic*. The Access Grid also uses a centralised server that implements a virtual meeting room metaphor called *Virtual Venues*. Meetings are held in a specific Virtual Venue and are joined by 'entering' the appropriate venue, which automatically launches the correct audio and video streams. Each Access Grid node typically transmits video streams in parallel from four remote controlled cameras, meaning that each person in the conferencing room is usually covered by at least one camera. Incoming video is projected on the wall of the node by several video projectors, which can display dozens of incoming video windows simultaneously. This means that everybody at remote sites can have continuous "presence" in a session. Loudspeakers and tabletop microphones are used in conjunction with echo-cancellation hardware to enable the Access Grid to support natural hands-free voice communications. Desktop versions of the Access Grid are also available, which allow users without access to a room-based node to participate in meetings from a PC.

A technician known as a *node operator* is normally present for each Access Grid session. Their job is to operate the software and hardware, performing such tasks as joining the correct Virtual Venue, controlling the local cameras and selecting which incoming video feeds are displayed. Node operators at each site use a text-based MOO (Object Oriented MUD) for back channel communications, allowing them, for example, to coordinate any technical adjustments without disrupting the meeting. The Access Grid also uses software called Distributed PowerPoint (DPPT) to enable a presenter to display and control a slide show at multiple sites from a laptop PC.

3.9.2 Access Grid Weaknesses

The Access Grid is often used for large meetings. For example, certain regular management meetings in the UK involve in the order of 12 nodes and have over 25 participants, and it is not unusual for other meetings that involve fewer nodes to have up to a dozen participants at each site. The author has a large amount of first hand experience of such meetings, both as a participant and as a node operator, and along with other participants has found that keeping track of who is in the session and identifying who is speaking can be difficult tasks that can be highly distracting from the meeting content. Figure 3.1 shows an actual screenshot of what is displayed to participants on the projection wall during a typical large Access Grid session. This screenshot clearly shows that participants can be overwhelmed by the amount of visual information they are presented with, making it difficult to determine who is currently in the session or who is currently speaking. To make matters worse, it can be made even harder to keep track of who is in the remote meeting rooms because not everybody is always on camera or displayed on the projection wall.

3.9.2.1 Other Technologies

Some of the Access Grid weaknesses described here are also present in other videoconferencing technologies. In particular, all the discussed shortcomings of Access Grid would most likely be shared by any large-scale continuous presence videoconferencing system that used a comparable number of video feeds.

Other videoconferencing technologies (e.g. H.323, H.320) use a single voice switched video stream that is distributed between multiple sites using a Multipoint Control Unit (MCU). While only viewing a single voice switched stream solves the problem of participants being overwhelmed by sheer number of video feeds, it does not help

keeping track of who is in the remote meeting rooms, as only a subset of participants will be visible at any one time. This means that the Access Grid weakness of not being able to keep track of participants still holds true (and could actually be worse for) MCU-based technologies.

Additionally, in a large scale audioconferencing environment, the Access Grid shortcomings for speaker identification and keeping track of who is in the session also may hold true. This is because there is no visual information to aid these basic tasks.

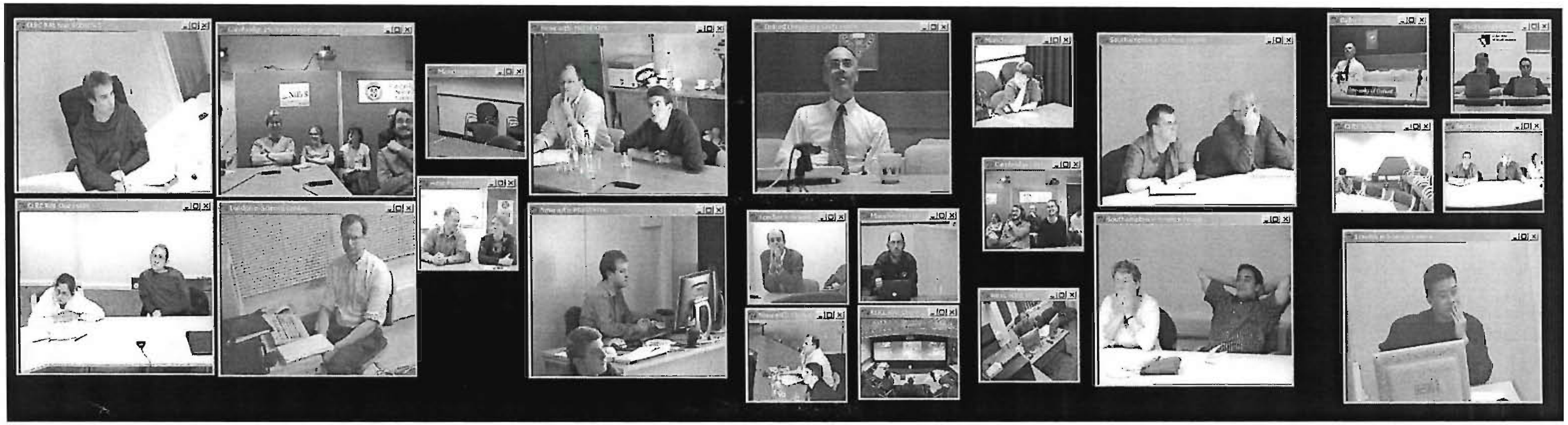


Figure 3.1, A screenshot of the Access Grid projection wall.

It is important to note that weaknesses described here mainly apply to large scale collaboration sessions, since identifying who is speaking and keeping track of who is in a session is usually simple when there are only a small number of participants.

3.9.3 Enhancing the Access Grid with Semantic Annotation

This section describes a fictional scenario where semantic annotations are used to enhance Access Grid videoconferencing. The annotation types include speaker identification and participant tracking to address the weaknesses identified in the previous section. The scenario also shows how the annotations fit in with other emerging services on the Semantic Web such as calendar scheduling and Communities of Practice.

This scenario makes the assumption that there is a queriable Semantic Web infrastructure in place (such as that described by De Roure and Hendler, see section 2.3) that allows information to be retrieved about specific resource instances. It also assumes that every meeting participant carries their own iButton for personal identification. iButtons [iBu04] are a form of contact memory that can be read by pressing them into a suitable reader. Each iButton contains a chip with a unique 64bit identifier and the overall package is about the same size as a house key.

The scenario describes a hypothetical first meeting between employees on a new project consisting of 15 people distributed across 4 different sites.

Project leader Tom would like hold an initial all hands project meeting over Access Grid. He instructs his calendar agent (see section 2.4.4) to book a meeting for all members of the project. It automatically assigns the meeting a unique URI and arranges a mutually agreeable date with all the other calendar agents of the project members and the calendar agents that handle the bookings for their local Access Grid nodes.

The day before the meeting, project member Alice finds out at short notice that she has another important meeting to go to at the same time as the project meeting. This new meeting is unavoidable so she will have to go to that one instead. She amends her online diary accordingly.

We now follow the events in one of the Access Grid nodes on the day of the project meeting. Before the meeting is underway, participants are shown a list of names on the projection wall of those participants already present at the remote sites and a list of people still expected. Importantly, this list does not include Alice, so the participants know its fine to start the meeting without her.

Participants identify themselves to the system by signing into the meeting using their personal iButtons in a readers located at their seating positions. In order to map the iButton ID to each person, the system queries the Semantic Web to resolve the iButton to its owner. Each owner of an iButton is responsible for publishing this information about their own iButton. This means that the system doesn't have to maintain this knowledge, and it is not limited to a closed set of users. For example, Tom is hosting a visitor on the day of the meeting and invites her join their meeting. The system is able to query the Semantic Web and retrieve her iButton information and uniquely identify her, even though she is not a project member and was not scheduled to turn up.

As each participant joins the meeting, they are notified via their laptops of any people in the meeting who have any indirect links to them that they might not be aware of (such Communities of Practice were discussed in section 2.4.1.4). The notifications not only identify who is in the Community of Practice, but also how they are related. This may in turn help shape the current collaboration or foster future collaborations by exposing hidden links between people, such as shared work interests.

All scheduled participants have arrived and the meeting is now underway. The list of participants and sites is displayed on the projection wall. Whenever a participant speaks, their name is highlighted in the list. Additionally, the border of the video window(s) originating from the site that the speaker is at are highlighted while they are speaking. This would not only aid with identifying who is speaking, but could help 'put a name to a face', which might be helpful in situations like this where the participants are not familiar with each other.

As the meeting progresses, Tom is gets distracted. He is using his laptop to reply to an urgent email. Gavin has a particular point that he wishes to address to the group and especially Tom. However due to the number of participants, it is not obvious to Gavin that Tom is distracted and if Gavin makes his remark now it will be wasted. Fortunately an annotation is displayed on the main projection wall stating that Tom is currently distracted and Gavin is able to save his remarks for later when Tom again has his full attention focused on the meeting.

John now has a slide presentation to deliver to the group. Initially the presentation goes well, but he has misjudged the level of technical depth to go into and the rest of the group rapidly become uninterested. John doesn't notice this as its not obvious through the video mediated communication and he is focused on delivering his presentation. However, an annotation is delivered to John's laptop that pops up informing him that the group are currently uninterested. John quickly realises that he has gone into far too much detail and has bored the group. He continues with his presentation, but provides fewer details and he is able to recapture the interest of the group. This is confirmed to him by another annotation stating that the group are interested again.

After John's presentation and as the meeting draws to the end of the allocated time, there is a sudden flurry of highly productive conversation led by Tom. The participants are engrossed and don't realise that the meeting is about to overrun. The Access Grid rooms are not booked for use by anybody else, so this is not an immediate problem for the meeting. Tom however has another meeting scheduled after this one and is in danger of running late.

The system displays an annotation to the group showing that Tom is due to be attending another meeting. The participants see this annotation and are able to start to wrap up their discussions. Unfortunately, they don't manage to wrap things up in time, and Tom is now late for his next meeting. Fortunately the system identifies this and instructs the other meeting room Tom is scheduled to be at to display an annotation stating that Tom is going to be late.

At the end of meeting the system automatically emails out a web link to each of the project members and to Tom's visitor. When this link is opened in a web browser it launches a fully indexed replay of the meeting.

Later on, Tom uses this link to easily locate and replay all the sections of the meeting where he was distracted. Alice, who was unable to attend, uses it to catch up on what happened in the meeting too. Unfortunately she is busy and doesn't have much time.

She particularly needed to see John's presentation, so she jumps straight to the section of the meeting where John was the main speaker.

3.9.3.1 Discussion of scenario

Behind the scenes, each of the annotation types presented here were generated through inferences. The annotations about the participants being present were obtained through combining separate facts about an iButton being docked, the owner of that iButton and the location of the iButton reader. These three facts were then combined through inference to assert that there was a specific identified person present in that meeting room.

Similarly, the speaker identification used inference to combine separate facts about audio levels on the microphones, the locations of the microphones and the locations of the participants. This inference is described in more detail in section 5.3.1.

If we imagine that there was an agent running on Tom's laptop monitoring the use of applications on the laptop, the inference about Tom being distracted was made by combining the facts about him currently being present in the meeting, him (rather than anybody else) being logged into that computer and the email program being used on that computer.

The group's level of interest could potentially be gauged through (mostly yet to be developed) computer vision techniques examining the body language of participants. The individual extracted body langue cues of participants could be combined to make the higher level assertion that the group as a whole is disinterested. If vision based techniques seem somewhat far off, a simpler solution would be to have a software interface running on the laptop of each participant with buttons next to categories that allow them to explicitly convey their current mental state. Again, each of these individual contributions could be combined through inference to determine the overall state of the group.

This scenario has not only shown the advantages of annotations, but has also shown how through using the Semantic Web, these annotations can be linked in with other services. For example the annotations about people being present was used in conjunction with the calendar information to infer a list of people still scheduled to arrive. Similarly, the system was able to uniquely identify Tom and determine that he was due to be in another meeting. An inference based on Tom's current location and his diary information enabled an annotation to be generated stating that he was due to be at the other meeting.

Here we see the benefit of the Semantic Web's ability to uniquely identify resources across domains and their relationships to other resources, which in turn can be used with inference to combine multiple facts to make meaningful assertions.

Chapter 5 describes an implementation based on a subset of the functionality presented in this scenario. This subset is limited to the speaker identification and participant tracking functions. Despite being based on a subset of this scenario, the implementation still demonstrates the general purpose infrastructure required to combine knowledge from multiple sources in real-time using inference. To avoid later disappointment, it should also be noted here that the window highlighting functionality of the scenario is not implemented either, as this would have proven to mainly be an exercise in modifying the video tool used by the Access Grid (vic).

3.10 Summary

This chapter has provided motivation for the addition of semantic annotations to live collaboration. Annotations can be used to present useful additional information to session participants, which could provide useful benefits such as increased awareness amongst participants. The annotations can also be used to index recordings of sessions and be used to provide a more complete replay than audio and video alone would provide. It has been shown that the semantic based approach to annotation provides excellent scope for inference, interoperability, reuse and extensibility, which promotes automation and reduces maintenance effort.

The study of W3C telephone conferences provided strong evidence that live annotation of collaboration sessions is useful to real users and this study aided the creation of a list of example useful annotations. Finally a scenario was presented in which it was described how dynamically updated attendance lists and real-time speaker identification could be used to overcome some of the shortcomings of Access Grid and other conferencing technologies.

4 A Framework for Real-Time Semantic Annotation

This chapter presents an event based framework for the automatic semantic annotation of distributed real-time collaboration activities. The framework is described in generic terms and consists of producers and consumers, which communicate using a shared tuple space. An inference engine coupled to an external triplestore is used to automatically infer further events from events directly captured from a collaboration session. This chapter does not discuss the design of the ontologies; this is instead discussed in detail in chapter 5.

4.1 Framework Origins

This section sets out to explain the thinking behind the conceptual architecture for the framework. The design decisions and specific technologies for this framework are then discussed in detail in the remainder of this chapter.

From the discussion in the previous chapter, it is possible to see that the framework needs to generate annotations that are triggered by events that occur during collaboration sessions. Therefore components are required that feed into the system, capturing events from the real world and generating a description of those events. Here these source components are called *producers*.

It is also required that annotations be displayed to participants during collaboration sessions. Hence some data sinks are required to receive annotations and display them to session participants. Here these sinks are referred to as *consumers*.

So far then, there are events being captured by producers, which are converted into annotations, which are transported to consumers for display to participants. However, as stated in the previous chapter, the intention is to use inference to obtain further annotations from existing annotations. The following section describes how inference can be incorporated into this model.

4.1.1 Inference

There needs to be some form of inference component or components that receive annotations from the producers, perform inference on those annotations and then make any new inferred annotations available to the consumers. Note that the inference component(s) should not prevent consumers from still receiving basic, non-inferred events from producers too. That is, the inference component(s) should not block these annotations, as the consumers may wish to still receive them.

There are several candidate places where the inference function could take place:

- At each producer.
- At each consumer.
- At each collaborating site.
- At a single centralised location.

There is no obvious advantage to having multiple inference components and it would result in needless replication of functionality and would complicate the architecture. Therefore the author feels that a single, centralised inference component (an inference engine) provides the neatest architecture. Furthermore administration, such as keeping the inference logic up to date would be easier.

This centralised component receives all annotations generated by producers in a collaboration session, performs inferences on those annotations, and makes any new inferred annotations available to the session consumers. In fact the inference component acts as both a consumer (receiving annotations) and a producer (generating new annotations). It is also likely that to make useful inferences from annotations, the inference process may need to access further knowledge from a repository similar to the CS AKTiveSpace triplestore.

The inference engine has access to all the annotations within a collaboration session (both those generated by producers and those it generates itself). As it is the only component that has access to all of these annotations, it makes sense that this component should also be responsible for placing the session annotations into persistent storage so that they may be archived.

4.1.2 Storage

The previous chapter argued that annotations from a session should be archived to create a record of the session that could potentially be later replayed in synchronisation with recorded audio and video.

To achieve this, there needs to be some form of persistent storage in the framework. A key issue here is whether there should be multiple components that provide this storage or just a single, centralised component. Multiple components would most likely prove to be more scalable and fault tolerant, but could make it difficult to locate specific items of knowledge. Hence the decision here is for a single, centralised store (i.e. a triplestore) as that would lead to a less complex framework. Experience from CS AKTiveSpace has shown that a centralised store can perform well for even for relatively large scale applications.

In addition to the storage needed for archiving annotations, storage would also be required for any additional knowledge that might be required to feed into the inference process. Consumers in general may also need access to further knowledge in addition to the annotations they receive to allow them to display meaningful human readable information (e.g. to resolve a URI to a human readable name). This knowledge could be held in separate stores, but there would be no real reason for segregating this knowledge. It makes more sense to hold this knowledge in the centralised triplestore along with the archived annotation data. Not only does this simplify the architecture, but also means that knowledge about previous collaboration sessions could then be easily used in the inference process if required.

4.1.3 Communications

The core framework components (i.e. producers, consumers, inference engine and triplestore) have now been discussed. What has yet to be discussed are the communications between these components.

Since the sites that makeup a collaboration session often change from session to session and individual sites are free to add new producers or consumers, it would not be practical for every consumer to know the location (e.g. IP address or DNS name) of every producer, or vice versa. Furthermore there could be many producers and many consumers, and it would also not be practical for explicit communications channels to exist between every producer and every consumer. Instead, a wiser solution is to use a publish and subscribe (pub/sub) model, in which components communicate via some intermediary, without needing to be explicitly aware of the existence of each other. This means that producers publish their annotations and consumers subscribe to only the annotations they require.

This pub/sub model also works for the producer and consumer functionality of the inference engine. The engine can subscribe to all the annotations it requires and then publish any new annotations it infers.

The requirements for the communications between producers and consumers are discussed in more detail in section 4.4, where it is also shown that the requirements are well met by a tuple space communications model.

Although this pub/sub model fits well with the producer/consumer architecture, it is not suitable for making the knowledge in the triplestore available. As the triplestore could potentially contain a large number of triples, it would be impractical to publish every single one of these. Instead, a standard query and response mode of communication is more practical here. The triplestore is also expected to remain at a fixed location, so it is reasonable that each component that needs to query it be pre-configured with its location.

Uploading annotations to the triplestore for archiving could be achieved by making the triplestore subscribe to all the annotations from a collaboration session. However, such behaviour is not a standard feature of existing triplestore implementations. Furthermore there appears to be no particular merit to doing this, hence a more standard approach is adopted where the inference engine explicitly uploads annotations to the triplestore for archival.

4.2 Overview of Framework

Figure 4.1 shows an overview of the semantic annotation framework. Basic meeting events are captured by producers and are encoded as RDF based annotations. These annotations are packaged as tuples and are published to a tuple space bound to the collaboration session.

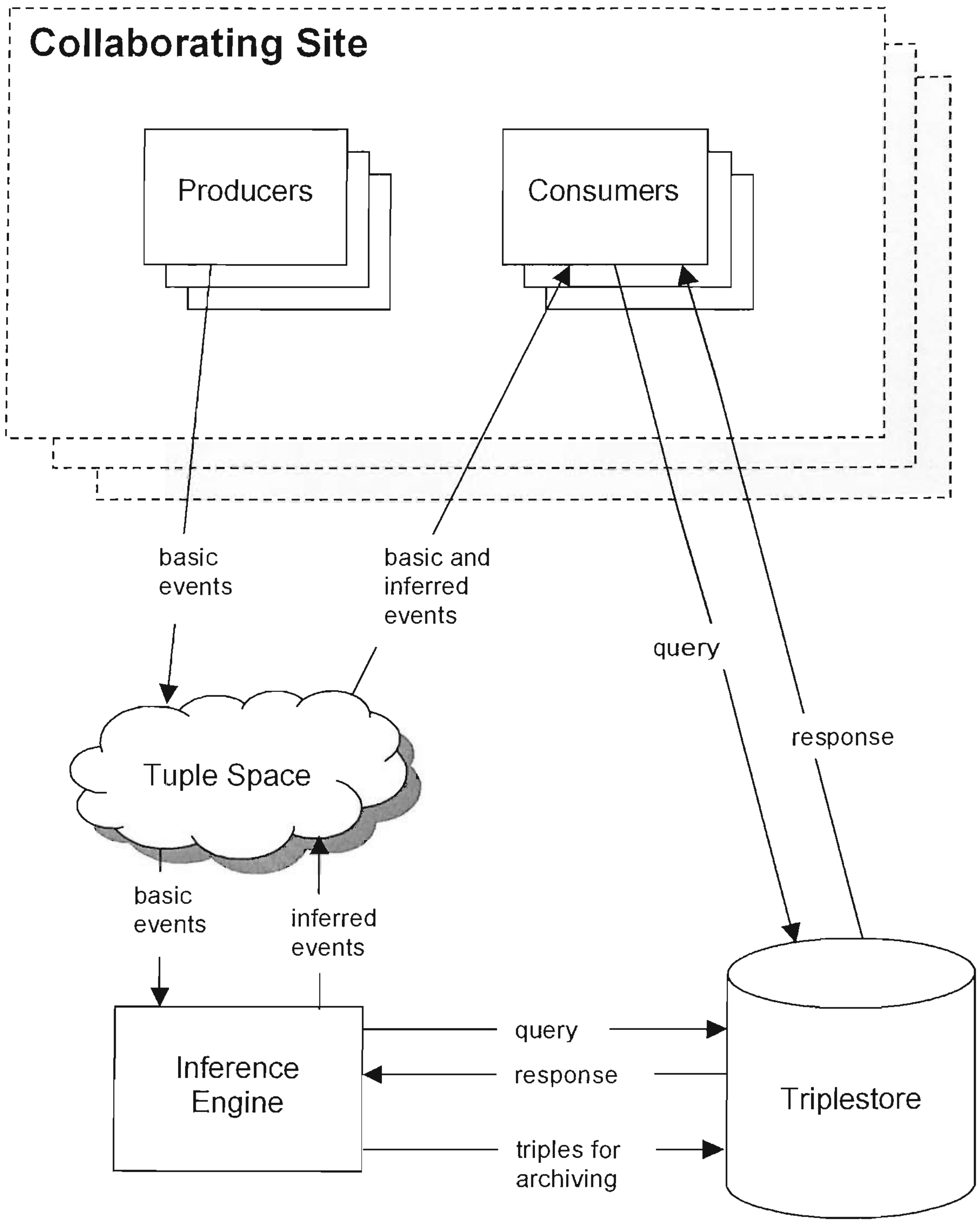


Figure 4.1, An Overview of The Semantic Annotation Framework.

An inference engine is also joined to the tuple space and subscribes to events generated by the producers. The inference engine has a number of domain specific inference rules, which it uses in conjunction with the external triplestore to infer higher level events from the basic events captured from the session. These higher-level events are also published to the tuple space to be used as annotations.

Consumer applications are also joined to the tuple space and subscribe to specific event types. One possible role of consumers is to display events to session participants in a human friendly form.

The inference engine is also responsible for storing the triples that describe each event (both basic and inferred) in the external triplestore. This provides a permanent semantic record of the collaboration session, which can be used in combination with an audiovisual recording to index and replay the session.

4.2.1 Comparison to Real-Time Expert Systems

This architecture presented here has some similarities to real-time expert systems based around the blackboard architecture that was popular in the 1980s (a good introduction is provided by [Cor91]). This section briefly compares the architecture presented here to blackboard based approaches.

In blackboard systems, there is a shared working memory called the blackboard into which input data (e.g. from sensors) is placed. The problem solving knowledge come from a collection of specialised components called knowledge sources, each of which is able to solve one particular aspect of the problem in hand and contribute to the information in the blackboard, providing incremental solution generation. As one knowledge source contributes to the information in the blackboard, this may in turn provide other knowledge sources with sufficient information to start solving their specific aspect of the problem they have knowledge about. Each knowledge source is treated as an independent 'black box' that performs a complex function.

Blackboard systems are event based, and knowledge sources subscribe only the specific event types (i.e. changes to the blackboard) that they are able to handle. Events can be triggered by changes made by knowledge sources or by external event sources (e.g. sensor input). To ensure efficient use of knowledge sources and prevent them attempting to access the blackboard all at once, there is a single controller component that determines the most appropriate knowledge source to execute in response to any particular event.

In the semantic annotation architecture presented in this chapter, there are a number of similarities to a blackboard architecture. The tuple space represents the shared working space and there are producers (analogous to sensors in the previous blackboard example). We also have event based subscriptions that ensure the inference engine only receives the event types it can handle. The main difference however is that there is a single inference engine rather than the multiple knowledge sources of the blackboard approach. The level of granularity in the semantic annotation framework is at the level of individual rules in the inference engine, rather than entire knowledge sources. This removes the need for the controller component present in blackboard systems and also means that changes to the inference process can be made through simply modifying rules, rather than modifying entire knowledge source components. Another difference is that the semantic annotation framework explicitly incorporates a long term persistent storage component (the triplestore) which provides bootstrapping knowledge and archives the inferences. Such a component is not a standard part of the blackboard architecture.

4.3 Events

The previous chapter discussed events during collaboration activities, and such events are the base concept in semantic annotation of these activities. This section discusses the representation of these events in this framework.

Events during a collaboration activity are discrete entities and need to be generated and transported as such. Hence some form of discrete packet should be used to represent these events. Each packet contains a payload that describes the event and is valid for a limited time interval, meaning that the packet representing an event must record this time interval. Incorporating a pair of timestamps into each packet may seem like the obvious solution. However, for live collaboration each packet needs to be transported at the beginning of each event to be of use, meaning that the end time is unknown at that moment. Therefore in the general case, events are represented by the two state changes that go to make up each single event, i.e. when the event starts, the state of the collaboration activity changes from that event not occurring, to the event occurring, and when the event ends, then the state changes back to that event not occurring. These two state changes are represented as two state change packets, each containing a *single* timestamp. The first packet is created and transported at the start of the event, and when the event ends a second packet is created and transported to indicate that the event has finished.

4.4 Event Sharing

The mechanism for sharing event state change packets between producers and consumers must support a number of features to be suitable for this framework. These features are:

- **Pub/Sub.** As discussed in section 4.1.3, in general it is not feasible for all consumers to directly communicate with the producers, hence a pub/sub model supporting indirect communication is more practical.

- **Real-time**. The time taken for a state change packet to be transported from a producer to all consumers should be small enough to be perceived by framework users as being sufficiently immediate.

- **Reliable**. In general, users of the framework will not tolerate lost or corrupt events. State change packets should be delivered without error and be guaranteed to reach all subscribing consumers.

- **Multipoint**. In general, there will be multiple components needing to share events simultaneously.

- **Persistent state changes for duration of session.** See next paragraph.

Each state change packet should persist in the sharing mechanism for the duration of the collaboration session. This allows any late joining consumer to be able to determine the current meeting state, even though it wasn't present when the state changes that describe the current meeting state were initially published. Making all state changes persistent allows consumers to obtain the full history of the session, which may be useful for presentation to participants, or replay applications. In many cases making all state changes persistent may in fact be overkill and impractical, as any late joining consumer could potentially be flooded with many state changes, past and present when they first join. This could be a particular problem for lightweight consumers. Where a full history of events is not required, a better solution is to only make state change packets persistent if they represent currently active events and remove them once they are no longer active.

This framework assumes best effort Quality of Service (QoS) and because of that there is some degree of conflict between the first two requirements. In particular, reliable delivery is not compatible with true real-time delivery since if network congestion or an error is encountered it requires that a packet is re-sent which introduces additional delay. In this framework it is anticipated that users will find a slight additional delay preferable to events being dropped. For example a slide transition that is slightly delayed is preferable to it not being delivered at all.

4.4.1 Tuple Spaces

These requirements for sharing events map well to a tuple space. Tuple spaces were pioneered in the Linda system [Car89] developed at Yale University in the 1980s. A tuple space can be thought of as a shared buffer that can contain tuples. In general, a tuple is simply a list of values, and is often used as a key-value pair.

Tuple spaces allow distributed components to communicate without being aware of the existence of each other. Components can publish tuples in the space or subscribe to be notified whenever tuples matching a subscription are published or modified. Tuple spaces are a form of associative memory, with tuples being accessed by matching some or all of the elements to values presented in a template. The template is just another tuple created for specifically for the purpose of matching.

Alternatives that were considered for providing the event sharing in the framework were content based routing, such as Elvin [Seg00] and a reliable multicast framework, such as Scalable Reliable Multicast [Flo97]. The main drawback to both of these approaches is that they do not support any form of persistence, which means that a late joining consumer would not be able to easily determine the current state of the collaboration session. An additional drawback to using a multicast based framework is that it would not support subscriptions to events of specific types, but would instead deliver all event to all consumers, which could potentially put significant network and processing load on the consumers, therefore excluding the use of lightweight consumers.

Although a tuple space fulfils the requirements of pub/sub, reliable, multipoint communications, and session level persistence, tuple spaces aren't usually associated with the real-time domain. There is however no reason why real-time tuple spaces cannot exist. Indeed, the implementation discussed in Chapter 5, uses a third party implementation of a real-time tuple space called EQUIP [Gre02].

4.4.2 State Change Packets

Each state change packet is represented as a single tuple. Each tuple contains a pair of values, which can be thought of a key-value pair. The first value is the type of event that this state change describes. This is represented as a URI and enables consumers to subscribe only to tuples that represent specific event types. The second value in the tuple is the full serialisation of the RDF triples that represent the state change the packet is describing. Subscriptions to a specific event type are achieved by consumers specifying a tuple with a URI of the event type as the first value and a 'wildcard' as the second value.

4.5 Real-Time Considerations

Since this framework needs to support real-time collaboration, some consideration needs to be given to what is meant by 'real-time' in this context. Depending on the application area, real-time can be either be a term used to describe a system that responds within a small and specified period, or a term whose definition is couched in terms of human perception, being a level of responsiveness that a user senses as sufficiently immediate.

Here the author uses the second definition to define real-time in the context of this framework, since a system that responds within a specified period is vastly more complex to design and implement than one that does not have these guarantees. In fact, in this framework, there are no guarantees that the responsiveness will always be sensed by users to be sufficiently immediate, as there may be times when noticeable delays may be perceived, for example as a result of a complex inference. It is for this reason that the author favours the term *near real-time*, rather than just *real-time* to accurately describe this framework. Miller [Mil68] found that users interacting with systems could tolerate response times of up to one second and still perceive the system as interactive. Therefore, the author defines the meaning of 'near real-time' in this thesis to be a response time of one second or less. Since the term 'near real-time' is somewhat clumsy to write, whenever this framework is described as 'real-time', what it is actually meant is that it is 'near real-time'.

It is worth noting that the amount of delay users may tolerate might depend on the event type. The different level of tolerable delay for each event type is not clear and is an area for further study. For example, for a speaker identification event, only a delay of a few hundred milliseconds might be tolerable, whereas for a sign-in event, a delay of several seconds might be tolerable.

4.6 Synchronisation of Events and Media Streams

Ideally this framework would be able to explicitly synchronise the display of events and corresponding media streams at each collaborating site. Unfortunately, as will be explained in this section, there appears to be no way to do this in the general case. It will be argued that explicit synchronisation, although preferable, is not required by this framework to support real-time collaboration.

Synchronising events with the media streams means that a consumer at a given site will start to display an event when the corresponding time point in the media is reached at that site (or at least within a small, fixed time interval bounding this point), e.g. if an event has a starting timestamp of x, then a consumer at a specific site should display that event when the corresponding time point x in the media streams is displayed to the participants at that site. Note that at each site, the display of the received media streams and events will always lag behind the actual current wall clock time because of encoding, decoding and network transit delays. E.g. a frame in live video that was taken at wall clock time y, will actually get displayed to participants at a remote site at wall clock time y + d, where d is the sum of the encoding, decoding and network transit delays, even though the frame being displayed represents wall clock time y.

The main reason that explicit synchronisation is not possible is that packets that make up the media streams must be delivered using an unreliable protocol, and that each media packet has a fixed deadline by which it must arrive in order to be displayed. Any media packet arriving after this deadline is dropped. However, events take time to be generated, either directly by a producer or by the inference process. This time may be difficult to predict and, when the events are inferred, may be significantly larger than the time taken for the corresponding point in the media streams to be generated. Furthermore, as discussed in the previous section, events are delivered using a reliable protocol, and events that take a longer time to arrive are not normally dropped. All this means that there is no deadline by which events arrive, and there is no way of making the media streams 'wait' to preserve synchronisation with an event that takes a longer time to arrive. When the media stream is analogue voice telephony, a similar argument holds true, even if the underlying mechanisms are different.

This problem could be fixed to some extent by introducing some additional buffering of the events and media streams at the receiver, but as the time taken for events to arrive is potentially unbounded, then some finite buffering would not be always guaranteed to fix the problem. In fact any significant buffering would introduce a far worse problem by destroying the interactive nature of the system, which is vital for real-time collaboration.

Instead, synchronisation in this framework relies on the real-time nature of the mechanisms used to capture or infer, transport and display the events, and to encode, transport and display the media streams. If this is done in real-time, then the events and media streams will be presented to users in near synchronisation. Although the synchronisation is implicit, it should be sufficient to be perceived by humans as being synchronised. An added bonus of not explicitly synchronising events and media is that it means the events can be totally independent of the technology chosen to encode and transport the media streams.

As mentioned in section 4.3, each state change packet contains a timestamp. However, as shown here, such timestamps are of limited use during live collaboration, since each state change packet is normally presented to participants as soon at it arrives at a consumer. As will be mentioned in the next section, timestamps do have a use in live sessions for performing sanity checking and determining things like network transit times.

The primary purpose of timestamps is in fact for the accurate archiving and replay of collaboration sessions. If producer generated timestamps were not used, then an archiving consumer would have to timestamp state change packets as it received them. This would be inaccurate because the timestamps would include the time taken to generate the packet and also the network transit time. For this reason it is better to use timestamps generated by producers.

4.7 Timestamp Generation and Format

This framework has different distributed producers generating state change packets with timestamps. Clearly there needs to be some common shared time between components, otherwise the timestamps may be inaccurate relative to each other. Therefore all components in the framework must be time synchronised to a common clock. Clock synchronisation may also be required for accurate synchronisation of multiple media streams during archiving.

The most sensible common time to use is UTC (Coordinated Universal Time, formerly known as GMT). Time synchronisation can easily be achieved by running a standard NTP (Network Time Protocol) [Mil92] client on each component, which is capable of synchronisation typically with an accuracy in the order of a few milliseconds, which should be sufficient accuracy for most conceivable applications of this framework. Forcing all components to use UTC makes the framework independent of local time zones, which otherwise would complicate matters during sessions in which the constituent sites spanned multiple time zones.

The framework uses absolute timestamps. One possible format for such timestamps could be an integer that records the time as a number of milliseconds since midnight UTC January 1st 1970, which is the format in which most computer systems record their time, and many programming languages provide functions to obtain this value directly. This is by no means the only suitable format for absolute timestamps and in the implementation discussed in chapter 5, a different but equivalent format is used for compatibility with an existing ontology.

The alternative to using absolute timestamps would be to use timestamps relative to the start of the collaboration session. The drawback of using relative timestamps is that each site joined to the collaboration session would need to have a shared knowledge of when the session started, which might not always be the case. Furthermore, using absolute timestamps means that each timestamp records the full date and time at which each event occurred, which provides more information than a relative timestamp.

In the case where the media streams are being recorded, the component recording the media streams also needs to record the UTC time it started recording the media streams to enable synchronisation of events and the media streams during replay. When replaying an archived meeting, it can use this as an offset to synchronise the archived state change packets with the media streams.

Having both producers and consumers synchronised to a common clock means that the network transit time for each non-inferred state change packet can easily be determined. This is achieved by calculating the difference between the timestamp and the consumer's current clock time. For inferred events, this calculated time would also include the time taken for the inference process, since an inferred event would most likely reuse the timestamp from the event it was inferred from. In either case, if the calculated difference was found to be excessive, then this could be flagged to an operator for further investigation. The most likely causes would be network congestion or excessive processor load on the inference engine. If a negative difference was calculated, or a very large positive value (i.e. much greater than any plausible network transit time and rule firing time), then it would most likely mean that either the producer or consumer, or even both, were not synchronised to UTC.

4.8 Detailed Description of Framework Components

This section completes the chapter by giving a detailed description of the framework components, drawing on the discussion in the previous sections. Figure 4.2 shows the complete set of framework components.

4.8.1 Producers

Producers are typically simple devices that capture basic meeting events. These devices can be lightweight and embedded and have very simple, one off, configuration requirements. Typically the only configuration a device will need is its location

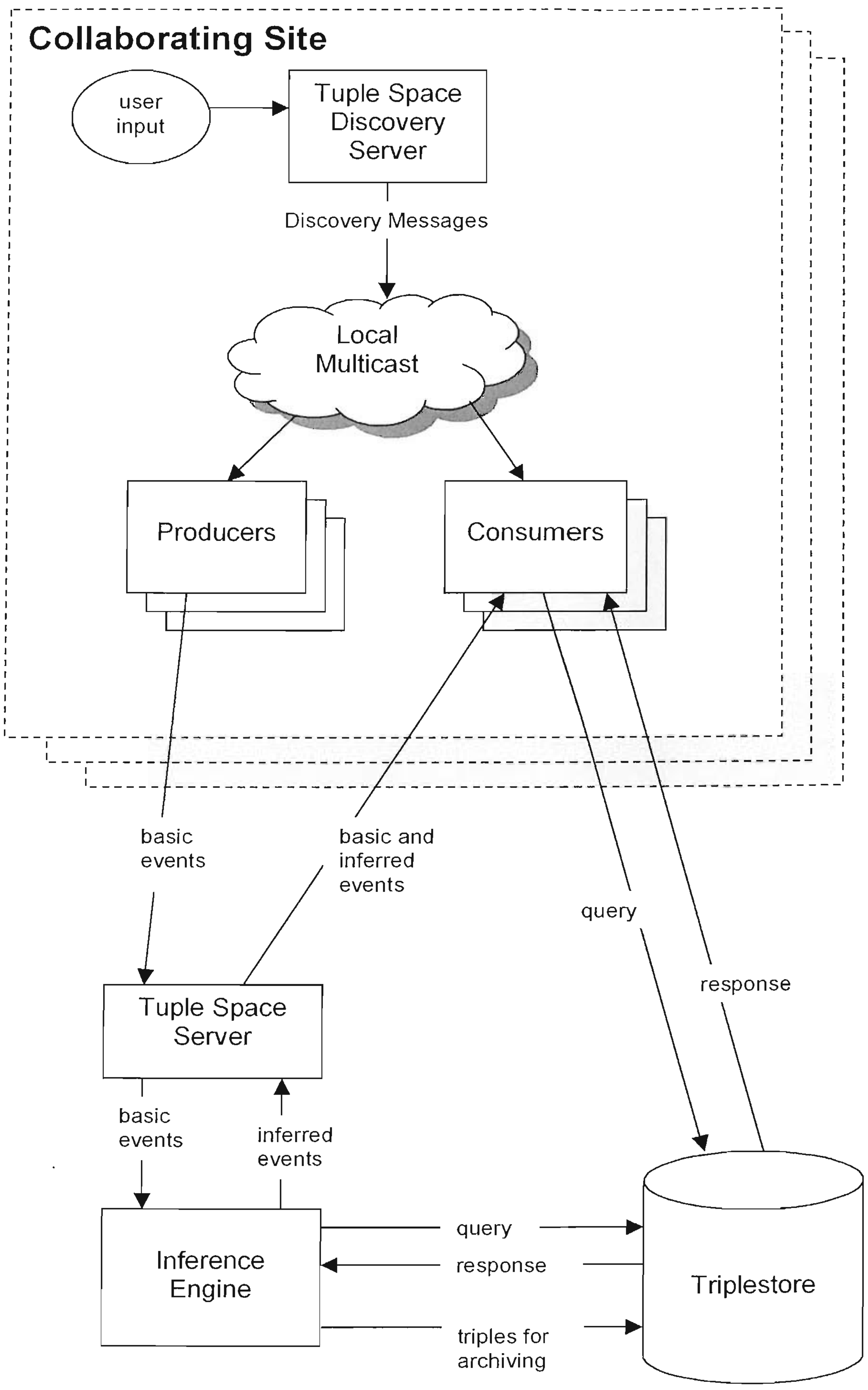


Figure 4.2, The Framework Components

(represented as single URI), and a multicast address and port on which it listens for tuple space discovery announcements (see section 4.8.3).

Each device is programmed before deployment to generate state change packets containing RDF triples describing the events they capture. The device location is used to feed into the triple generation process so that the location of each event is specified, which is potentially useful for the inference process. The triples generated by producers conform to a pre-shared OWL ontology, so that they can be understood by consumers, and in particular the inference engine. Producers are also responsible for taking each

batch of triples they generate that describes a state change and packaging them as a tuple and publishing them to the tuple space bound to the current collaboration session. An example of a producer could be a digital audio mixer used to generate events that describe microphone activity (from which events describing who is speaking could be inferred). Such a specialist device is likely to need to be connected to a host PC to carry out the functions of generating the triples and publishing them to the tuple space.

4.8.2 Tuple Space Server

The tuple space server is responsible for implementing the shared buffer that provides the tuple space functionality. Producers and consumers may connect to this server to access the tuple space. Each collaboration session requires an instance of a tuple space to be running on the server.

4.8.3 Tuple Space Discovery Server

It would not be practical to manually instruct each producer and consumer at each collaboration site of the tuple space bound to each collaboration session. For this reason, this framework uses a dynamic tuple space discovery mechanism, the principles of which are taken directly from the EQUIP tuple space implementation [Gre02]. The discovery mechanism works by having each collaboration site run a tuple space discovery server. The discovery server sends out discovery messages every few seconds on a site local multicast group. The address and port is known to each producer and consumer at the site as it is included as part of their initial configuration, and each producer and consumer at a given site subscribes to this multicast group.

Whenever a collaboration session starts, as part of session establishment, the person operating the session at each site inputs the tuple space parameters into the discovery server (a tuple space discovery server should have a user interface to achieve this). This server then broadcasts them as discovery messages. Each discovery message identifies the tuple space bound to the current collaboration session. On receipt of such a message, all producers and consumers at the site join the tuple space specified by the parameters.

When the collaboration session ends, the operator inputs into server that the session has ended. At this point, each message multicast from the server instructs each producer and consumer to disconnect from the tuple space.

Since multicast is unreliable, the potential exists that a discovery message may not reach all producers and consumer at a site. This is not a problem, since the server sends out repeat messages every few seconds, so if a message is lost, another one will be broadcast a few seconds later.

In an environment where multicast is not available, dynamic discovery can be achieved by having a tuple space server at each collaboration site, which runs a default discovery tuple space. Producers and consumers join the default discovery tuple space and discovery messages are published as tuples.

4.8.4 Inference Engine

For each collaboration session there runs an instance of a forward chaining rule-based inference engine. This is joined to the tuple space for the collaboration session and acts as both a consumer and producer, subscribing to basic meeting events and inferring higher level events from them and publishing these to the tuple space.

Each instance of the inference engine is joined to a specific instance of the tuple space, and this joining persists between individual collaboration sessions. This means that the inference engine does not have to dynamically discover the tuple space for each collaboration session, but instead can be told this in a one off configuration step. The number of instances of the inference engine and the tuple space running at the same time determines how many collaboration sessions may take place in parallel.

The inference engine is pre-configured to subscribe to all the different known non-inferred event types generated by the producers. It adds the triples from these events into its own internal triplestore, which represents its knowledge of the current collaboration session. (To avoid confusion with external triplestores, this internal triplestore shall be referred to as a knowledgebase.)

The inference engine has pre-authored, domain specific inference rules that fire in real-time when a matching pattern of triples is found in its knowledge base. These rules then generate triples that represent higher-level events. These triples are added to its knowledge base, and as with other producers, are also packaged as tuples and published to the tuple space for the benefit of other consumers. In addition to using pure rules based inference, the inference engine also queries an external triplestore whenever it has gaps in its knowledge that prevent it from making a specific inference.

For example, in the implementation described in the next chapter, one of the inferences is that there is a person present in a specific seating position of a specific meeting room. This inference is made as the result of a number of rules firing after there is an iButton sign in event. The presence of the sign in event in the knowledgebase triggers a rule that queries the triplestore for the seating position and room location of the iButton reader. This new information in turn triggers another rule that queries the triplestore to determine the person who owns the iButton. The presence of this ownership and location information in the knowledgebase then causes another rule to fire which combines this knowledge to infer that the particular person is present at that seating position of that meeting room. Full details of the inference process are described in chapter 6.

The inference engine is also responsible for archiving events to the external triplestore. This is a sensible choice because at the end of a collaboration session its knowledgebase contains the complete set of triples that represent all the events from the collaboration session, both basic and inferred. Since an inference engine is not suited for persistent storage, all the triples in its knowledgebase are transferred to the external triplestore once the collaboration session has finished. All triples are then deleted from the knowledge base, leaving the inference engine ready for a new collaboration session.

Once the triples representing a collaboration session are added to the external triplestore, these could then be accessed by replay clients through suitable queries.

4.8.5 External Triplestore

In essence the external triplestore is a persistent repository of knowledge, represented as triples that may be queried and added to as required. It is difficult to generalise about what kinds of knowledge the triplestore must contain to be useful, as it is largely domain specific. In the implementation described in the next chapter, the triplestore is predominantly used for looking up where components are located in a location hierarchy, as this was found to lead to useful inferences. E.g. if an iButton reader is located in a specific seating location, then it is reasonable to infer that the person who just signed into that reader is sitting in that seating location. The triplestore is also queried by consumers to resolve URIs to human readable names where required for display purposes, so it is likely that the triplestore will need to contain this type of knowledge too.

How the triplestore is initialised and maintained is beyond the scope of the framework, but it is possible that automated techniques such as those used in CS AKTiveSpace (see section 2.4.1) could be used to extract knowledge from existing sources, in conjunction with some hand authoring of information not already available in an electronic form.

This framework assumes that there is a single triplestore and that it remains at a fixed location. All components that need to query the triplestore (i.e. the inference engine and consumers) are given the location of the triplestore as part of their initial configuration. The specific mechanism for querying the triplestore is not defined by this framework.

4.8.6 Consumers

Consumers are typically some form of application for displaying events in a human understandable form. The application may be interactive, for example displaying information in the form of hyperlinks. Consumers, however, do not necessarily have to be applications responsible for display. For example, the inference engine acts a consumer too.

Consumers subscribe to the specific event type(s) that they are able to handle, and parse the serialised RDF from received state change packets into triples. In the case of a display application, these triples are converted into a human understandable form and displayed in real-time. Part of the consumer's process of converting triples into a human understandable form may include queries to the triplestore to resolve URIs into a human readable form.

Consumers may run on devices such as a standard PC, wireless PDA or even a mobile phone. In general consumers may be more complex than producers, since they have to perform more tasks, including parsing RDF, user interface functions and queries to the triplestore. Furthermore, a consumer may need to handle multiple event types, which further adds to the complexity.

4.9 Summary

This chapter has presented a generic framework for the automatic semantic annotation of distributed real-time collaboration activities. Each semantic annotation is an event represented as two state changes, allowing live creation and transport of events. It has been explained that explicit synchronisation of annotations with the media streams would be difficult in the general case, and that implicit synchronisation is sufficient for real-time collaboration.

The framework consists of producers and consumers that communicate using a shared tuple space. This enables reliable, loosely coupled pub/sub communication and also supports late joining consumers. An inference engine that exhibits both producer and consumer functionality is used to automatically infer further collaboration events from those captured by other producers. This inference engine uses an external triplestore as a source of additional knowledge that feeds into the inference process.

5 Implementation

This chapter describes a proof of concept implementation of the framework presented in the previous chapter. The annotation functionality of the implementation is based on the Access Grid scenario from chapter 3. An ontology is presented that represents concepts such as events, time, locations and people, a significant portion of which was reused from existing ontologies. A third party inference engine was used and a number of inference rules were created. In addition to this, three producers were created for capturing and publishing events from collaboration sessions. A single consumer was also created for displaying attendance lists and speaker identification data.

5.1 Overview of End-User Functionality

The functionality chosen for implementation was based on a subset of that described in the scenario presented in section 3.9. This subset was real-time speaker identification and dynamically updated attendance lists.

The implemented system described in this chapter also differs from the scenario by not implementing video window highlighting. Window highlighting was not implemented as it was decided that this would prove to be mainly an exercise in modifying vic, and would not have much relevance to the semantic annotation framework. A replay function was also omitted from the implementation since this was a feature that had already been provided by the CoAKTinG meeting replay tool (see section 2.2.7). Furthermore, at the time of writing the author is employed on a project called Memetic [Mem05] developing the CoAKTinG meeting replay tool to support automated and semi automated annotation of recorded Access Grid meetings. Inference will be one of the techniques used in Memetic for achieving this.

The implemented system is presented to participants at each site as a dynamically updated list of sites and names of current participants. This list is displayed on the main projection screen at each site in the session. The list consists of a number of headings, each one being the name of a site that is currently part of the Access Grid session. As sites join or leave, appropriate site headings are automatically added or removed. The purpose of the site headings are so that all participants know exactly which sites are in the session at any particular moment. Below each heading is a list of the names of each participant at that specific site. Entries in this list are automatically added or removed as individual participants join or leave the meeting. When a participant speaks, their entry in the list is highlighted to indicate they are speaking. This list is intended to increase participant awareness by explicitly listing session participants and to make it easier to identify who is speaking. The system is also able to determine when a meeting is in session at each individual Access Grid room, which could for example, be used to display the status of the Access Grid node on a screen outside the room.

From the perspective of the participants, all they need to do extra to achieve this functionality is to carry around a personal iButton and use this to sign into a reader which is located in every seating position. Behind the scenes, the system automatically generates an RDF description of each session, which is then archived in a triplestore.

Although the implementation is based around an Access Grid scenario, it is fairly general purpose and could equally be applied to group-to-group telephone audio conferences or other group-to-group conferencing technologies.

5.2 Overview of System

A diagram of the overall system architecture is shown in Figure 5.1. The entire system was implemented in Java and an overview of some of these components is given in the following sections.

5.2.1 Producers

The system required each meeting room to be equipped with the following producers:

- **iButton Reader Producer**. There needed to be an iButton reader located at each seating position, each of which was connected to a host PC at each site that was responsible for publishing annotations that described the sign in and sign out events at each iButton reader.

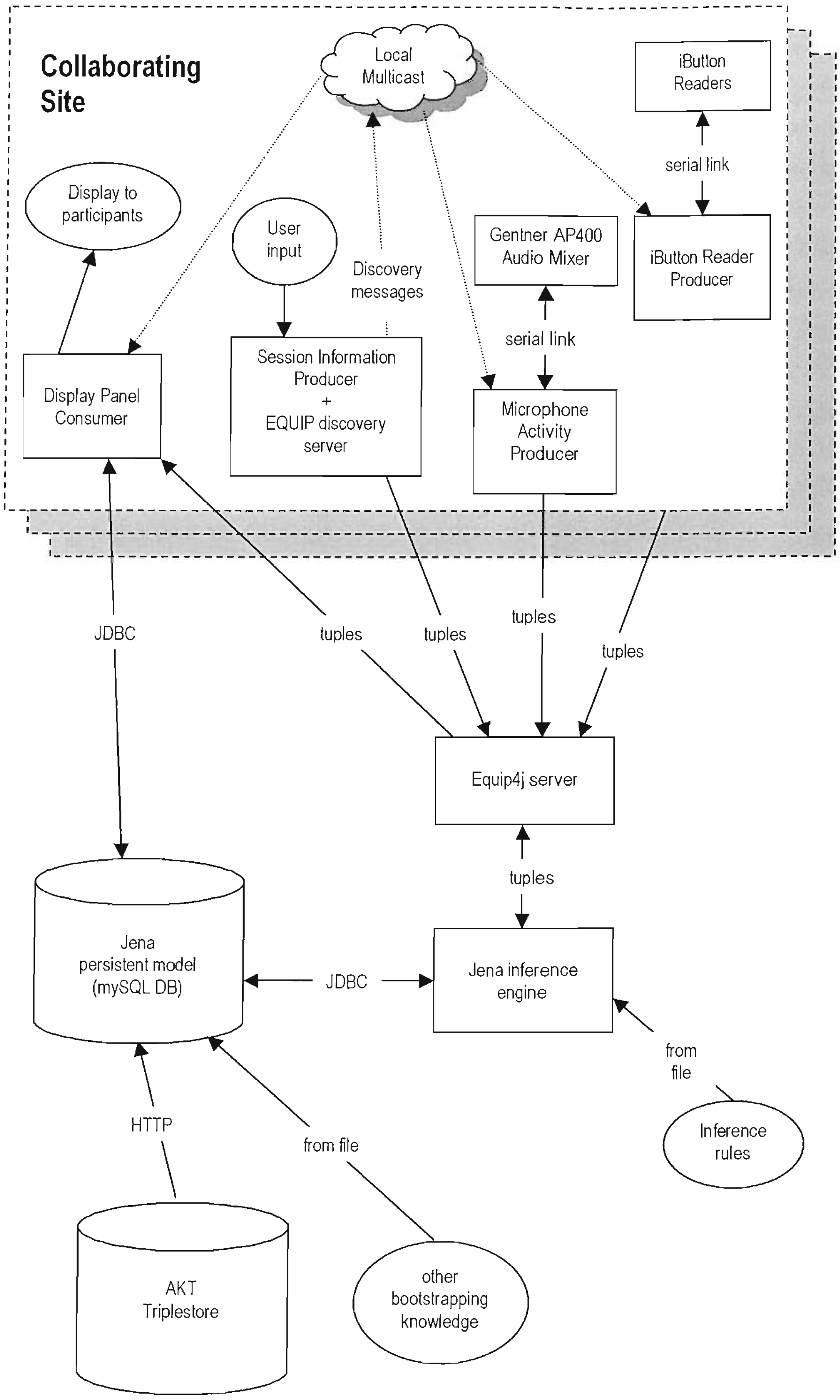


Figure 5.1, Overview of implemented system.

- **Microphone Activity Producer.** Individual microphones were located at each seating position, and these fed into a digital microphone mixer connected via a serial link to a host PC. The host PC was responsible for publishing annotations when sound was detected past a certain threshold at each individual microphone.

- **Session Information Producer**. The session information producer was responsible for publishing annotations that stated when the meeting room had joined an Access Grid session. This information was manually entered by the node operator, but this could have been achieved automatically by integrating it with the Access Grid session handling software. The session information producer also ran a tuple space discovery server, which enabled the other components to join the correct tuple space.

5.2.2 Inference Engine

Central to the system was the inference engine. There was a single instance shared in a session, which subscribed to the events generated by the producers. It had a number of forward chaining inference rules, which it used in conjunction with queries to an external triplestore to generate the participant list and the speaker identification annotations.

In order to allow the display application to generate a participant list, the inference engine took iButton reader events and firstly queried the triplestore to resolve the iButton ID from the event to the URI for the person who owned that iButton. It also performed a further query to determine the seating location in which the iButton reader was located. Once these queries had been made this caused a further rule to fire, which inferred that the person who owned the iButton must be present at that particular seating location, and an annotation to that effect was published to the tuple space.

The inference engine also inferred participant speaking events. It did this by receiving microphone activity events and querying the triplestore to determine which seating position the microphone was located in. Once the seating location had been determined, a further rule fired, which used knowledge about who was sitting in that seating location to infer that the person had made a verbal comment, and an annotation to that effect was published. The full details of the inference process are discussed at length in chapter 6.

5.2.3 Consumers

The system had just a single consumer application, an instance of which ran at each site. This was the display panel application that was responsible for displaying the list of sites and participants, and the speaker identification data.

In order to allow the display panel application to display a list of connected site names, it subscribed to the room-mapping events from the session information producers. On receiving each event, it queried the triplestore to obtain a human readable name for each collaboration site, and then displayed the site name in the list.

In order to display the participant list, the display application subscribed to the person present events generated by the inference engine. On receipt of a person present event the display application queried the triplestore to obtain a human readable name for the person that event referred to, and then displayed the name in the list.

In order to display the speaker identification events, the display application subscribed to the speaker identification events generated by the inference engine. On receipt of an event, it highlighted the name of the appropriate person in the list, and when the event ended, it removed the highlighting from that person's name.

5.3 Speaker Identification Technique

As speaker identification was an important aspect of the implementation, some consideration is given here for the chosen technique, which also proved to be somewhat novel.

Numerous techniques exist for text independent automatic speaker identification, consisting of either speech pattern matching (e.g. [Bet00]) or Sound Source Localisation (SSL) techniques (e.g. [Cut02]). The drawback of pattern matching techniques is that they require users to supply a sample of their voice in advance and when in use typically require at least one or two seconds of speech before being able to produce a result and are therefore are not suitable for real-time applications or short utterances. The techniques also cannot handle two people speaking at once, which is a common occurrence in meetings.

For this reason SSL techniques were favoured here, and are particularly suited to meeting rooms, as participants tend to stay seated in the same location for the duration of a meeting. One of the drawbacks of SSL is that it normally requires expensive microphone arrays. The author has however discovered that it is possible to do basic SSL by exploiting a feature of the standard audio hardware that the majority of Access Grid nodes use.

Each room-based Access Grid node is equipped with an echo-cancelling digital audio mixer, usually manufactured by Gentner. These Gentner products can be controlled and monitored via the serial port of a PC allowing real-time access to the audio levels and gating status of each microphone. When a microphone is gated *on*, it means that the level it has picked up is above some specified threshold. Likewise, when gated *off*, it means that the level is below a certain threshold.

Most Access Grid nodes use a Gentner AP400, which has four microphone inputs. Initially it was hoped to do SSL by looking at the relative audio levels on each of the microphones. Unfortunately after some experimentation it was found that hardware limitations in the Gentner unit meant that it was only possible to access the audio level one microphone at a time, with a 0.2 second delay between accessing each microphone, and that this amount of delay was too large to do SSL using this technique. The gating status of the microphones, however, proved to be more useful. The Gentner hardware allows the status of all four microphones to be reported simultaneously once every 0.2 seconds, which is sufficiently fast to be useful. Furthermore, by default the AP400 has a feature called *First Mic Priority* mode enabled, which has a useful side effect, explained as follows. Its main purpose is to help maintain good speech intelligibility by ensuring that only one microphone gates on when a person speaks. It achieves this by determining the audio levels received by all the microphones when the first microphone gates on and this audio level is then used as the ambient level for all other microphones. The useful side effect of this is that if each participant has a microphone in front of them, then only the microphone in front of the person speaking will gate on. This means that the gating status of the microphones accurately reflects who is speaking at any moment in time. The author has confirmed this experimentally. Additionally, if more than one person speaks at a time, then a microphone will gate on for each person speaking.

Performing SSL by giving each participant a tabletop microphone may sound obvious, but without *First Mic Priority* mode the technique is not reliable. The author conducted experiments with the mode switched off and found that usually two or more microphones gated on when only one person spoke and that increasing the gating threshold simply meant that quieter speakers were unable to gate their microphone on.

The advantage of performing SSL in this way is that it requires no additional hardware for Access Grid nodes. The main drawback is that the maximum number of participants is limited to the number of microphones, although similar Gentner products (e.g. AP800) deployed at other Access Grid nodes support up to eight microphones, and multiple Gentner units can be daisy chained to provide more microphone inputs. Another weakness is that sounds other than speech can gate a microphone on (e.g. a door slamming), but it is likely that in most sessions this would not occur frequently enough to cause significant generation of incorrect speaker identification data. This weakness could be overcome by using a technique to determine if an audio signal is speech or non-speech (e.g. [Tur02]).

5.3.1 The need for inference

At this stage it may appear as if the AP400 is performing *all* the functions required for full speaker identification. However, the only information the AP400 asserts is the identity of the microphone that is currently gating on. Clearly this information on its own does not identify the actual participant who spoke, as it says nothing about any participant. In fact the AP400 has absolutely no knowledge about the participants and no knowledge about where its microphones are located.

Inference is therefore required to take the basic microphone gating knowledge from the AP400 and combine this information with other external knowledge about the locations of the microphones and the locations of the participants. Only once all this knowledge is taken into account by the inference process is it possible to make the explicit assertion stating which participant is currently speaking. In the implementation described here, four facts are needed to make the inference about who is currently speaking:

1. The identity of the microphone that is currently gated on
2. The identity of the person signed into a specific iButton reader
3. The seating position of that iButton reader
4. The seating position of the microphone

It is possible to see that only the first of these four facts comes from AP400, with the rest coming from the iButton reader and the external triplestore. The full inference process is described in detail in chapter 6.

5.4 Ontology

The key first step in implementing a system within this framework was to author the ontology for the various components to use. One of the many benefits of taking an approach based around the Semantic Web was that it facilitated easy reuse of ontologies. For this implementation, it was chosen to reuse ontologies wherever possible as this would reduce implementation effort and promote interoperability with existing tools.

5.4.1 Ontologies Chosen for Reuse

There are numerous existing Semantic Web ontologies, meaning that some consideration needed to be given to the ontologies that would be reused to form the basis of the implementation here.

The implementation required representations of concepts such as events, time, locations and people. Appropriate ontologies for representing these were found to be the ontologies from the CoAKTinG and AKT projects (see sections 2.2.7 and 2.4.1). The CoAKTinG ontology was originally created for the offline semantic annotation of recordings of synchronous collaboration activities and already had representations for things such as distributed collaboration sessions and people speaking. This ontology in turn reused a number of elements from the AKT Support and Portal ontologies to provide a representation for entities such as events, time, locations and people. The CoAKTinG ontology is given in Appendix D.

The representation of locations in the AKT Portal ontology, was however rather basic and did not provide any means to define the relationship between locations or spaces.

For this reason an additional location ontology was reused from the Signage Project [Mil04] to represent locations and the interrelationships between them. The Signage location ontology is given in Appendix E. Its use of hierarchical spaces enabled useful inferences to be made. For example, through a hierarchy specifying devices located in seating positions and seating positions located in rooms, it is possible to infer when two devices are located in the same room or in the same seating position, even though these explicit relationships are not specified. For example, consider the following list of basic facts that describe part of a meeting room:

1. iButton reader A is located in seating position 1
2. iButton reader B is located in seating position 2
3. Microphone X is located in seating position 1
4. Seating position 1 is located in room Y
5. Seating position 2 is located in room Y

Its possible to see that in addition to the explicit facts, several further facts can be inferred. For example, fact 1 and 3 can be combined to infer that iButton reader A is co-located with microphone X in the same seating position. Facts 1, 2, 4 and 5 can be combined to infer that iButton readers A and B are located in the same room. As will be shown in chapter 6, these inferences are used as part of the process to determine when people are co-located or when people are sitting in front specific microphones, which are used respectively to infer when there is a meeting taking place and which participant is currently speaking.

Therefore using this representation and combining it with inference has removed the need to explicitly specify all the relationships between the locations. It also means that the configuration can be less application specific, for example another application could combine facts 4 and 5 to infer that seating positions 1 and 2 were in the same room. Furthermore, any changes to the device locations require only a single relationship to be modified, thus ensuring that the maintenance of the configuration is straightforward.

An additional advantage of using the AKT, CoAKTinG and Signage ontologies was that their creators were research colleagues of the author, which promoted discussion about issues such as design rationale and allowed the potential for input into future developments for these ontologies.

The ontologies described here were by no means the only suitable basis for this implementation. For example, the MINDSWAP conference ontology [Min04] has a representation of events, time, people, sub-events, attendees and location and the eBiquity Group [eBi04] have published similar ontologies too.

Since there was some reuse of existing ontologies in the implementation described in this chapter, a simple namespace mechanism will be used when describing ontology terms so the origin of each term is clear. Any term prefixed with the namespace 'live' (short for 'live collaboration ontology') is an original contribution by the author. All terms with other namespaces have been reused from existing ontologies. The live collaboration ontology is given in Appendix C.

5.4.2 Events and Time

Since events and time were key to the system, the discussion shall begin on this topic. It is fairly clear that a sensible way to structure the ontology would be to have a superclass representing a generic event and to subclass this event into specific event types. This is exactly what the CoAKTinG ontology does, taking a representation of an event from the AKT Portal ontology (referred to here as 'portal') and subclassing it into a number of specific event types.

Figure 5.2 shows the structure of the portal:Event class and the parts that it inherits from the AKT Support ontology (referred to here as 'support'). In essence, portal:Event is something that can have a beginning time, an end time, a location and any number of sub-events. An event can also have a 'main agent' and 'other agents involved' specified, which can, for example, be used respectively to specify the chair and other participants of a meeting.

portal:Event gets its ability to express a beginning time and an end time by sub classing support:Temporal-Thing, which possesses a support:has-time-interval property, which has a range of support:Time-Interval. support:Time-Interval represents a time interval by having a support:begins-at-time-point property and support:ends-at-time-point property, both of which have a range of support:Time-Point.

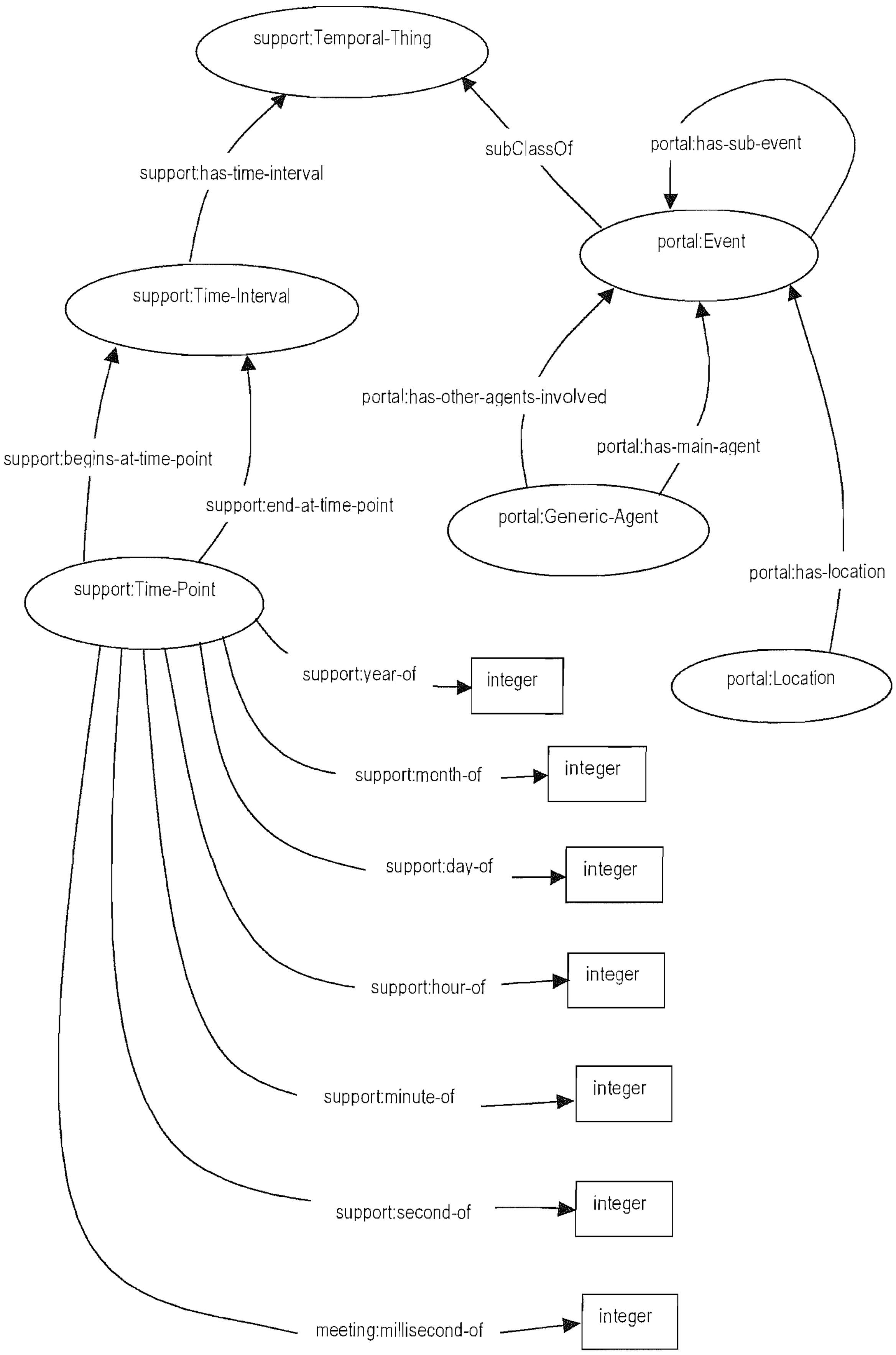


Figure 5.2, The portion of the AKT ontology representing events and time.

Although it may seem more intuitive to use two separate classes to represent an event in live collaboration (i.e. one to represent the beginning of an event and one to represent the end of the event), using portal:Event and only defining the end time once it is known achieves the same overall function. Furthermore, when archiving events, it is more intuitive having a single class representing an event.

It is for these reasons that the portal:Event was chosen as the superclass for all events in this implementation. The class required no further extensions to be suitable for use in this implementation.

5.4.3 Location

Location was also an important part of the ontology, as it played a key role in a number of inferences. The base concept of location was taken from two sources, the AKT Portal ontology and the Signage Project location ontology (namespace abbreviated to 'location' here), and was extended by the author to meet the specific needs of the proof of concept implementation. Figure 5.3 shows the location ontology used in the implementation.

In the Signage ontology, the most general type of location is a location:Abstract-Space. Specific types of location subclass this, such as location:Room and location:Work-Area. In the AKT portal ontology, the most general type of location is portal:Location, which is subclassed into specific location types such as portal:Country and portal:Geographical-Region. portal:Location is the range of the portal:has-location property of portal:Event.

So that events could use locations defined by the Signage location ontology, and to make the Signage concept of location interoperable with the AKT Portal concept of location in general, the author asserted in the live collaboration ontology that location:Abstract-Space was owl:equivalentClass to portal:Location.

From the names it may not seem that location:Abstract-Space and portal:Location were semantically equivalent concepts, so that declaring them as equivalent classes was not a valid thing to do. The author however argues that both classes are in fact semantically equivalent. In fact, in the English language the terms space and location are somewhat

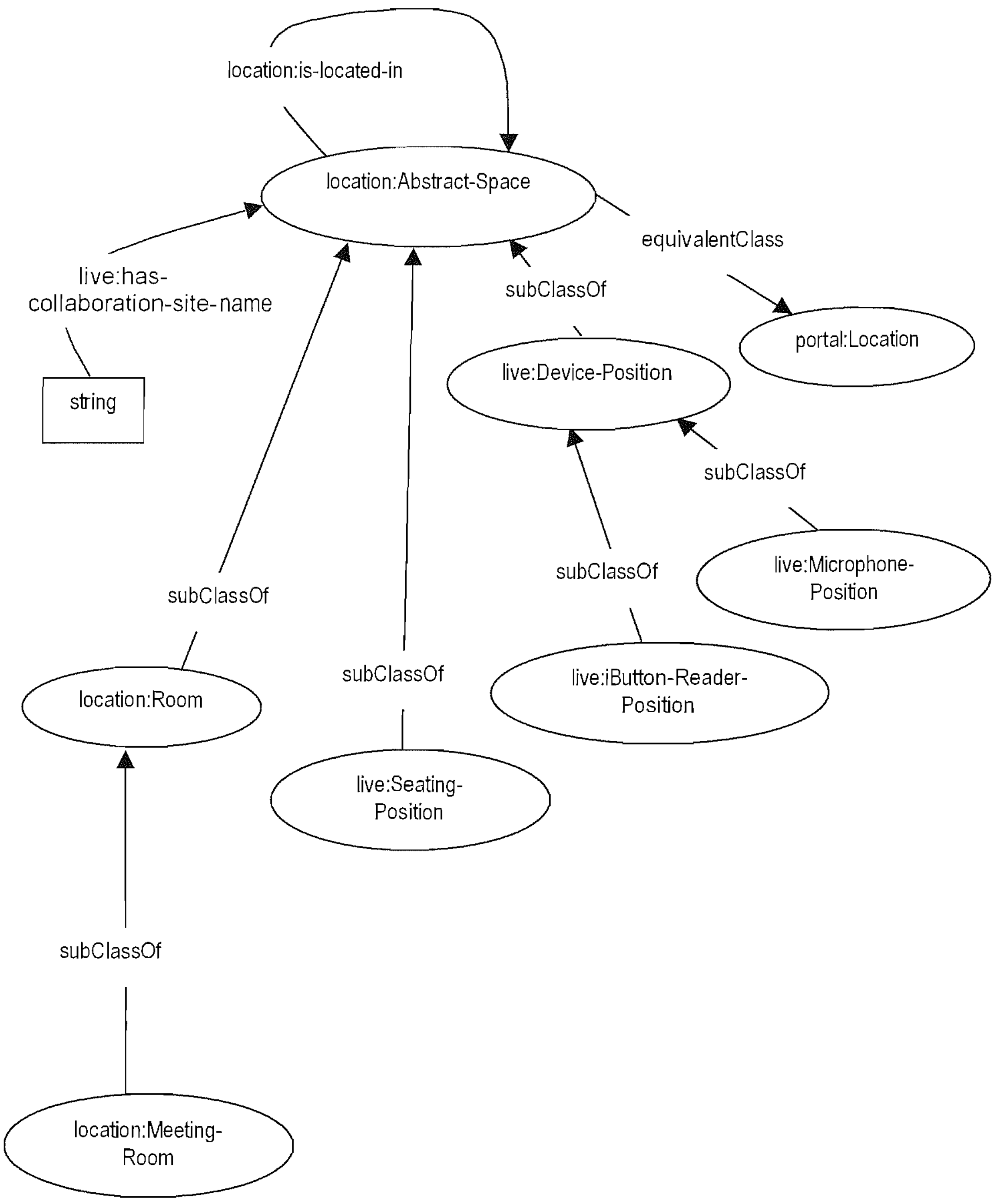


Figure 5.3, The location portion of the ontology

ambiguous, as a space can mean either a volume or area and a location can either be a specific point in space or a whole area such as a city (which arguably actually occupies a volume). This ambiguity is reflected in the Signage ontology by a location:Room and a location:Work-Area both being subclasses of location:Abstract-Space, while the former is arguably a volume and the latter is arguably an area. Similarly, although few specific subclasses of portal:Location exist, a room, or an entire country would both be valid subclasses.

To help overcome these ambiguities, the author defines a location:Abstract-Space and portal:Location as “any space, either in two or three dimensions where it is possible to define the boundaries (at any given moment of time)”. This definition should also be used whenever the author uses the terms ‘space’ or ‘location’. Note that although it is possible to define the boundaries of the space, these boundaries need not be explicitly defined somewhere in order to refer to that space, all that matters is that the boundaries *can* be defined. In the case of a specific point in space, this can be thought of as a bounded space of zero volume.

5.4.3.1 Location Types

The final ontology has knowledge of five different location types, these are a room, a meeting room, a seating position, an iButton reader position and a microphone position. All the location types are subclasses of location:Abstract-Space and are explained in this section.

The concept of room is taken directly from the Signage ontology and is represented by the class location:Room. This is used as a generic representation of any type of room. The concept of meeting room is also taken directly from the Signage ontology and is represented by the class location:Meeting-Room, which is a subclass of location:Room. This is used to represent any room whose primary purpose is for holding meetings, and was the representation chosen for Access Grid enabled rooms. It could be argued that the representation of an Access Grid enabled room should be represented by a class more specific than a generic meeting room (e.g. with a class like Access-Grid-Room), as this would give the potential for inferences that used knowledge that the room was Access Grid capable. While this is certainly true, the author feels that a generic meeting room is sufficient specialisation for this system. Furthermore, a better way of representing specific collaboration technologies available at that meeting room would be to represent them as properties of that meeting room, as that would allow a meeting room to support more than one type of collaboration technology, which is often the case.

Since the Signage and AKT ontologies had no representation of a seating position, iButton reader position or microphone position, these were added in the live collaboration ontology. A seating position is represented by the live:Seating-Position class. This is used to represent each individual seating location in a meeting room, i.e. the location occupied by a single meeting participant.

The final two new location classes defined were live:iButton-Reader-Position and live:Microphone-Position which are used to represent the locations the respective devices, i.e. the space occupied by the physical device. The classes both subclassed a generic class called live:Device-Position, which in turn subclassed location:Abstract-Space.

5.4.3.2 Abstract-Space Properties

The location:Abstract-Space class had two properties which were useful to this implementation. One of which was already defined in the Signage ontology and the other of which was a new property defined in the live collaboration ontology.

The first property was location:is-located-in, which had a domain and range of location:Abstract-Space. This is used to specify that one space is located in another space, for example a room being located in a building. In this implementation, this property was used to specify that a particular iButton reader position or microphone position is located in a particular seating position, and that a seating position is located in a particular meeting room. This property is clearly transitive, that is if, for example, microphone position A is located in seating position B, and seating position B is located in meeting room C, then A must also be located in C. At the time of implementation, the Signage ontology did not declare location:is-located-in as an owl:transitiveProperty, so this was rectified by declaring this property as transitive in the live collaboration ontology. This allowed precisely the kind of transitive location based inferences as described above. At the time of writing, the current version of the Signage ontology now incorporates this transitive property definition too.

The second property was live:has-collaboration-site-name which was a new property defined in the live collaboration ontology. This property was a literal string value and was used to represent a human readable name for an individual collaboration site, which for example may be used as the text for a site heading in the display panel application. This name should make sense in the context of an Access Grid (or other type collaboration session). An example of a name would be 'Southampton University, ECS'. The most likely subclass of location:Abstract-Space this property would be used with is a location:Meeting-Room, as that is the actual collaboration site. Note that this property does not provide a name for the meeting room itself, but for the site that the Access Grid node located in that room represents in an Access Grid session.

Rather that giving live:has-collaboration-site-name a domain of location:Abstract-Space, it may seem more appropriate to restrict its domain to location:Meeting-Room, as that is typically what is used to represent a collaboration site. Although that is the case for this implementation, doing so would remove the possibility of other spaces being sites of collaboration. For example, desktop versions of the access grid exist, so a desk could have a collaboration site name and although unlikely, its not out of the question for somewhere like a corridor or garden to be a collaboration site. By leaving the domain to be very general, this effectively allows any type of location:Abstract-Space to be a collaboration site.

5.4.4 Personal Identification

The portion of the ontology that relates to the identification of individual session participants is relatively simple and is shown in Figure 5.4.

The concept of person (i.e. the representation of people taking part in a collaboration session) is taken from the AKT portal ontology using the class portal:Person. The Portal ontology defines a number of properties on portal:Person, but the only one used by this implementation was portal:full-name, which specified the person's full name as a human readable string. This was used when displaying the names of participants in the display panel application.

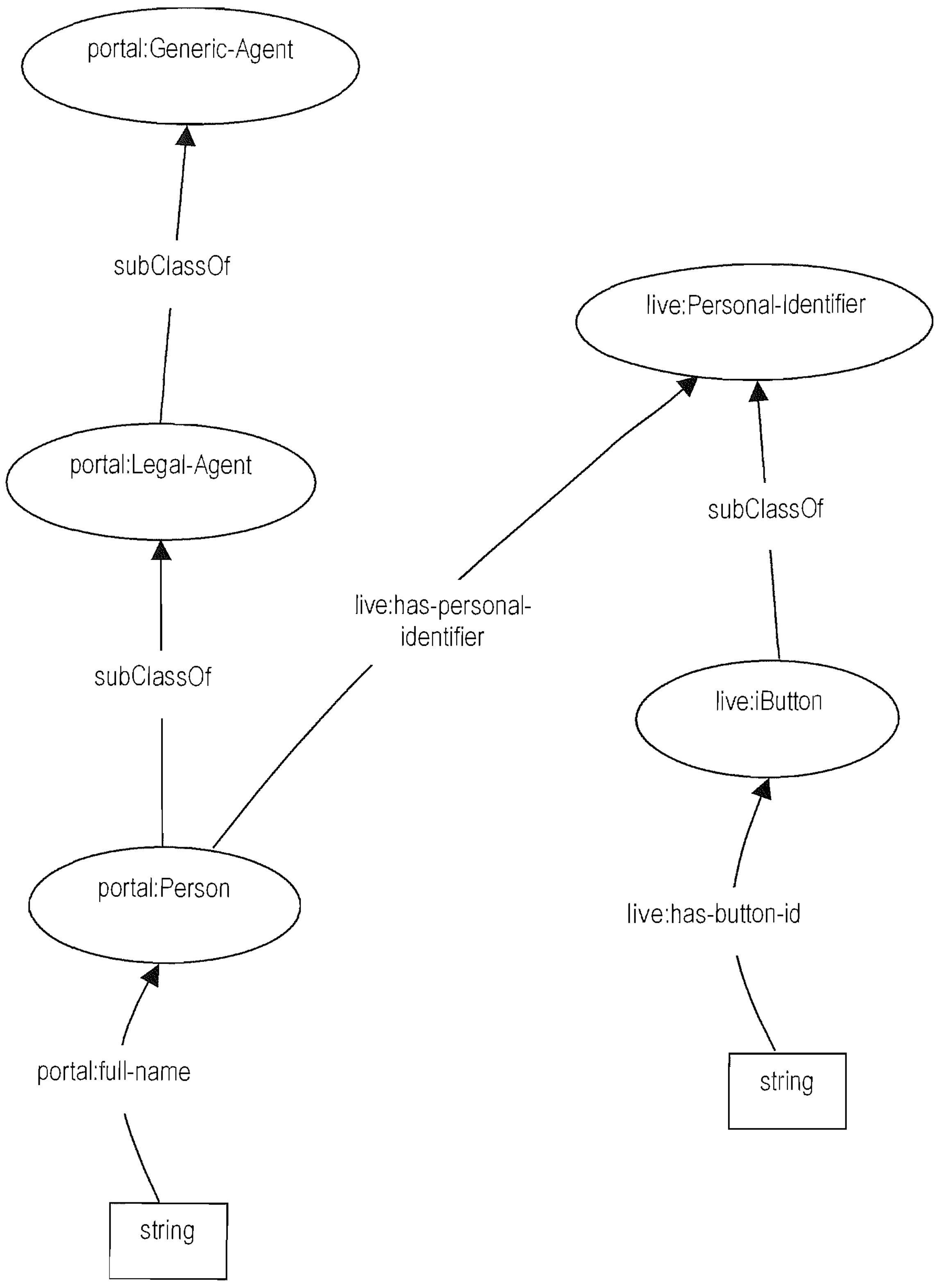


Figure 5.4, The section of the ontology for personal identification.

People identify themselves using iButtons, so these needed to be represented in the ontology. As no suitable representations were in the existing ontologies, a representation was added to the live collaboration ontology. This was achieved by firstly creating the generic superclass live:Personal-Identifier to represent all forms of tangible identifiers like iButtons that could be used to uniquely identify a person. This is then subclassed as live:iButton which is the class that represents an iButton. This class has a single literal string property live:has-button-id which actually records the unique ID of the iButton.

In order to tie an iButton (or any other personal identifier) to its owner, the live collaboration ontology also defined the property live:has-personal-identifier, which has a domain of portal:Person and a range of live:Personal-Identifier, making it possible to associate one or more instances of a live:Personal-Identifier with a person.

5.4.5 Event Types

With the rest of the elements of the ontology in place, the final part of the ontology is the portion that represents the specific event types used by the system. As with the other parts of the system, a number of the events were reused from existing ontologies. Figure 5.5 shows the portions of the event ontology reused from existing ontologies and Figure 5.6 shows the new portion of the ontology created specifically for the implementation.

5.4.5.1 Events From Existing Ontologies

A single event type is taken from the AKT portal ontology, namely portal:Meeting-Taking-Place. This is used to represent when co-located people are collaborating, such as the activity that takes place at each of the sites participating in an Access Grid session. The use of this class in this way (i.e. to represent the activity that takes place at each site during distributed collaboration) was first shown by the CoAKTinG ontology. A representation for a real-time distributed collaboration session is taken from meeting:Distributed-Gathering. This is used to tie together each of the individual portal:Meeting-Taking-Place events in order to form a representation of a distributed collaboration session. This is achieved by using the meeting:has-local-event property (which is a sub property of portal:has-sub-event), to specify each of the local meetings

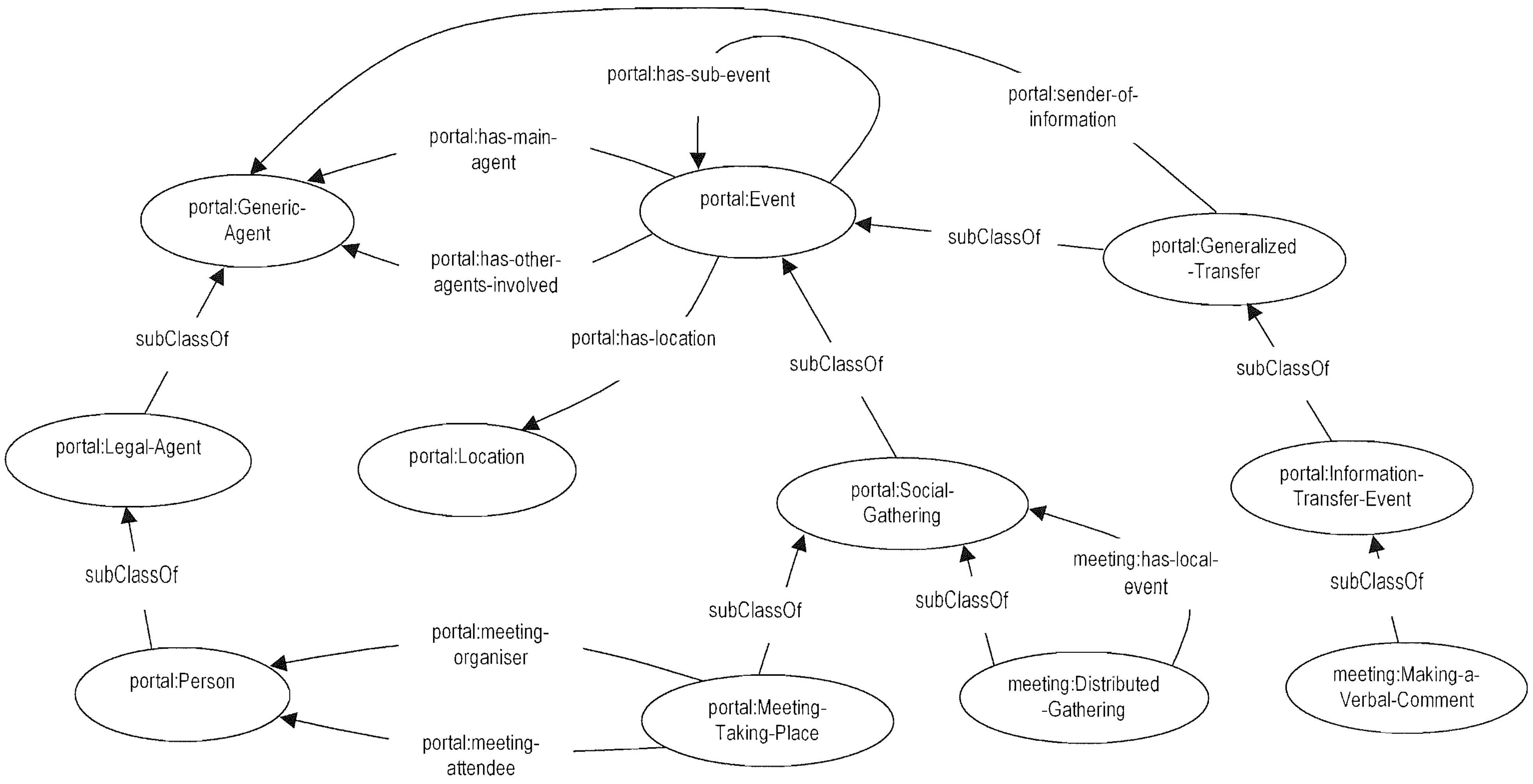


Figure 5.5, Event types reused from the AKT and CoAKTinG ontologies.

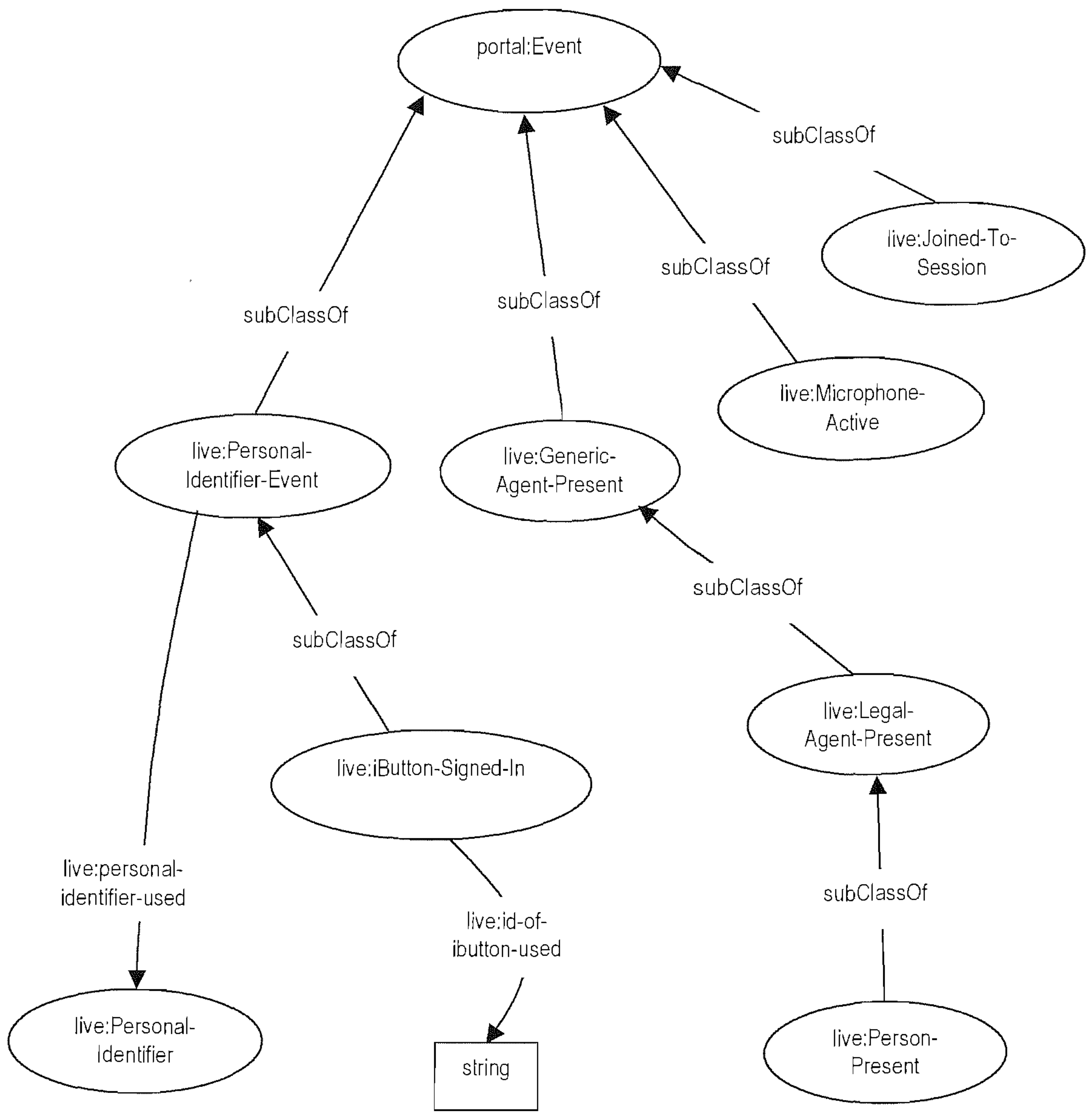


Figure 5.6, The ontology representing the new event types.

that make up the distributed gathering. The main purpose of these two events is to record who is at each site and which sites are in the collaboration session.

The CoAKTinG meeting ontology also has a representation for people speaking, meeting:Making-a-Verbal-Comment, and this was used for the same purpose in this implementation.

5.4.5.2 New Event Types

The first new event type required is to explicitly represent the notion of an individual meeting room being part of a collaboration session. This event is live:Joined-To-Session. The primary purpose of this event is during live sessions, to enable participants to tell when another site has joined the session, even when nobody has yet signed in at that site. It uses the inherited portal:has-location property to record the room location joined to the session. This event does not record any explicit session identifier, as it is implicit from the scope of the tuple space.

The next event type is used as a generic representation for any action involving a live:Personal-Identifier. This event is live:Personal-Identifier-Event, which is used as a base class for events such as people signing in or out using some form of personal identifier such as an iButton. The event defines a live:has-personal-identifier property which has a range of live:Personal-Identifier. This is used to specify the particular instance of a personal identifier used in the event. The event is subclassed by a class called live:iButton-Signed-In, which represents when an iButton is signed in. An live:iButton-Signed-In event begins when the iButton is signed in and ends when the iButton is signed out.

Since any device generating an live:iButton-Signed-In event would not normally have knowledge of the URI of the iButton that the event represents, an additional property called live:id-of-ibutton-used is defined for an live:iButton-Signed-In event. This is a literal value for recording the 64 bit iButton ID as a string of hexadecimal digits. This allows generation of these events without having to first determine the URI of the iButton.

The event of a person being present in a collaboration session is represented by the live:Person-Present class, which is an event that starts when the person joins the session (indicated by signing in with an iButton) and ends when the person leaves the session (by signing out). The person the event refers to is represented by the portal:has-main-agent property inherited from portal:Event. For a live:Person-Present event, this property has an owl:allValuesFrom restriction limiting it to be values from the class portal:Person.

Since in the AKT ontology, portal:Person is a subclass of portal:Legal-Agent, which in turn is a subclass of portal:Generic-Agent, it was chosen to mirror this class hierarchy above the live:Person-Present event. This meant defining a new class called live:Generic-Agent-Present, which was subclassed another new class called live:Legal-Agent-Present. The live: Person-Present class then subclassed live:Legal-Agent-Present. This was done to maximise the potential for any future interoperability with the AKT ontology.

The final new event type was live:Microphone-Active, which was used to represent when a microphone gates on. This event, along with live Person-Present is used as a basis for inferring meeting:Making-a-Verbal-Comment events.

5.5 Tuple Space

The tuple space was a key component of the implementation, as it provided the core communications service used by all the other components. The chosen tuple space implementation was EQUIP [Gre02], which was originally implemented for use in the EQUATOR Interdisciplinary Research Collaboration [EQU04]. As well as tuple space functionality, the full version of EQUIP also has a general event system and support for Collaborative Virtual Environments (CVEs), like Massive-3 [Gre00], with features such as 3D rendering. This full version of EQUIP has interfaces in both Java and C++.

Since much of the complexity of the full version of EQUIP was not required, the much simpler Equip4j [Equ04b] was used as the basis for the proof of concept system. It is a Java only subset of EQUIP that has a simpler mechanism for defining data items and does not support 3D rendering, which was not required anyway.

Numerous other tuple space implementations exist, many of which are also Java based, such as TSpaces [TSp04] and JavaSpaces[Jav04]. These would have been just as suitable for providing the tuple space functionality, but EQUIP was chosen as it was designed specifically for supporting real-time collaboration. Furthermore, the author, although not a project member, worked with EQUATOR researchers. This meant that there was potential for technical support if required and also for influencing future EQUIP developments.

The data sharing service in Equip4j, which is used to publish tuples is referred to as a *dataspace*. This is provided by *dataspace servers*, which are identified with URLs of the form "equip://host:port/name". Tuples are persistent for the lifetime of the dataspace, or until they are deleted by their owner. Equip4j also supports non-persistent tuples called tuple events. Both persistent and non-persistent tuples were used in the proof of concept implementation.

Each tuple consists of a number of ordered values, which are instances of the `equip.runtime.ValueBase` Java class, which is subclassed to be container classes for Java native types such as Strings, ints and arrays. Persistent tuples also have a unique identifier so they can be referenced by their owners, to allow functions like deletion.

Subscriptions to particular tuples are based on exactly matching the values held within the tuple against a template tuple. Wildcard values can be specified by using a Java null value. A convention in EQUIP is that the first value in the tuple should always be a pseudo class name for the tuple to say what type of tuple it is so, for example, different applications can share the same dataspace and only subscribe to the type of tuples intended for them. The EQUIP convention for a pseudo class name is a dotted hierarchical string such as "bpj00r.meeting.Event", although this syntax is not enforced.

5.5.1 Events as EQUIP Tuples

Each collaboration event is represented using two tuples, which represent the two individual state changes that make up the event. The state change representing the beginning of the event is published as a persistent tuple, while the end of the event is published as a non-persistent tuple event. Once an event has ended, the producer that generated the event deletes the beginning tuple for the event. This means that events persist in the dataspace for as long as they are still active, so that any late joining consumer is able to determine the current session state. There is no need to make the state change representing the end of the event persistent, since any consumer joining after the event has ended will have no knowledge of the event. In this implementation deleting events that are no longer active was preferable to making them persist for the duration of the collaboration session, as otherwise a late joining consumer could have potentially been swamped with inactive events when it first tried to join the session. There may be cases where consumers may need to display a history of the session, in which case making inactive events persistent is required, but this was not the case for the proof of concept implementation.

Each tuple used in the implementation contained three values:

- The first value was the pseudo tuple class name string, needed for EQUIP compliance. This was chosen to be "bpj00r.meeting.Event".

- The second value was a URI represented as a Java String. This URI specified the RDF type of the event that the tuple represented, e.g. for a iButton-Signed-In event the string was:
"http://www.ecs.soton.ac.uk/~bpj00r/ontologies/live-meeting-20040319-1# iButton-Signed-In"
This type URI was specified both for tuples that represent the beginning of an event and the end of an event.

- The third value in the tuple was another String that contained the full serialisation of all the RDF triples that described the state change that the tuple represented. This serialisation was in N-Triples [Bec01], since this was a very simple form for producers to generate directly without the need for using any external software libraries, as would otherwise have been the case if say full XML serialisation syntax had been used. For example, the full serialisation of an iButton-Signed-In event would be as follows: (note that the namespaces in this example have been abbreviated for readability)

```
myns:signinevent1 rdf:type live:iButton-Signed-In .
myns:signinevent1 support:has-time-interval myns:timeinterval1 .
myns:signinevent1 portal:has-location mylocs:sotonuni-B59-3241-seat1-reader1 .
mylocs:sotonuni-B59-3241-seat1-reader1 rdf:type live:iButton-Reader-Position .
myns:signinevent1 live:id-of-ibutton-used "02000009EA6FD301" .
myns:timeinterval1 rdf:type support:Time-Interval .
myns:timeinterval1 support:begins-at-time-point myns:timepoint1 .
myns:timepoint1 support:year-of "2004" .
myns:timepoint1 support:month-of "10" .
myns:timepoint1 support:day-of "26" .
myns:timepoint1 support:hour-of "21" .
myns:timepoint1 support:minute-of "6" .
myns:timepoint1 support:second-of "26" .
myns:timepoint1 meeting:millisecond-of "209" .
```

Explicitly stating the event type as a separate value in the tuples made it straightforward for consumers to subscribe only to tuples that represented the types of events they were able to handle. If this was not done, then the full serialisation of each event would have needed to be parsed to by consumers to determine the event type. Subscriptions therefore had the following template: the first value was always the string "bpj00r.meeting.Event" (as this is the EQUIP tuple type), the second value was the full URI of the event type the subscription was for and third value was always null to act as a wildcard that matched all RDF models.

5.5.2 Dataspace Discovery Mechanism

Equip4j also has a useful built in mechanism to allow dataspace clients (i.e. producers and consumers) to automatically discover specific dataspace instances. This discovery mechanism was used in the proof of concept implementation to enable producers and consumers to automatically join the correct dataspace for the current collaboration session.

The discovery mechanism works by having a discovery server running at each local network. The server sends out discovery messages to a predetermined local multicast group and each dataspace client is pre-configured to subscribe to this group. When the clients are required to connect to a particular dataspace, the server sends out a discovery message every few seconds informing the clients of the dataspace URL they should connect to. Similarly when the clients are required to disconnect, the server sends out messages informing the clients of this. How this discovery mechanism is used in this proof of concept application is discussed further in section 5.6.1.

5.6 Producers

Producers are responsible for capturing or inferring collaboration events and publishing them to the tuple space mapped to the session. A producer typically consists of some specialist hardware connected to a PC, or may be purely software based. Excluding the inference engine, the implementation used three different types of producers. These were: the Session Information Producer, the iButton Reader Producer and the Microphone Activity Producer. These are discussed in detail in the following sections.

5.6.1 Session Information Producer

A single instance of the session information producer runs at each collaborating site. This is a purely software-based producer whose purpose is to generate the live:Joined-To-Session events. The producer software is atypical in that it also runs the discovery server for the EQUIP dataspace.

The software runs as a basic command line application, which takes keyboard input directly from the Access Grid node operator. In order to instruct the producer that the Access Grid node has now joined the current Access Grid session (i.e. has entered the correct virtual venue for the meeting), the operator simply types in the URL that specifies the dataspace bound to the current session. This instructs the producer to join the dataspace specified by the URL, generate a Joined-To-Session event, and publish it to the dataspace as a persistent tuple. It also instructs the dataspace discovery server it runs to start multicasting discovery messages, instructing all the other producers and consumers at the site to join the correct dataspace for that session.

When the Access Grid session is over, the node operator instructs the producer of this by simply entering the string ‘end’. This causes the producer to delete the original tuple it published and to publish a further non-persistent tuple asserting an end-time on the Joined-To-Session event; the producer then leaves the dataspace. The discovery server then stops multicasting the discovery messages and starts multicasting messages instructing the other producers and consumers to leave the dataspace.

Clearly there is scope for automating this process with tighter integration with the Access Grid software (e.g. to automatically share the dataspace URL between sites and determine when the Access Grid session begins and ends), but this basic system was sufficient for the proof of concept system. It also had the advantage of being general purpose enough to be used with other collaboration technologies, such as telephone conferencing.

Each instance of this producer type has a very simple one off configuration, which is in the form of a text file. This file simply specifies the multicast group for dataspace discovery, and a URI which identifies the instance of Meeting-Room the Access Grid node is located in. This URI is used to provide the value for the has-location property of the Joined-To-Session event.

5.6.2 Microphone Activity Producer

The purpose of the microphone activity producer is to generate the Microphone-Active events that occur as a result of a participant speaking in front of a microphone. An instance of this producer runs at each collaborating site.

In terms of hardware, the producer consists of four tabletop microphones connected to a Gentner AP400 echo cancelling digital audio mixer. The AP400 is then connected to a PC using an RS232 link. Each microphone is positioned at a fixed seating location around a table in the meeting room. Since the AP400 supports a maximum of four microphones, so this limits the total number of seating locations to four also, although as discussed in section 5.3 other hardware solutions are available that support more microphones than this.

The producer software on the PC repeatedly polls the mixer for the gating status of the microphones. Each time it detects that a microphone's gating status has changed from *off* to *on*, it creates a Microphone-Active event, with the location property set to the Microphone-Position of the microphone that gated on. It then publishes this event to the dataspace as a persistent tuple. When the microphone gates off, the producer then deletes the previous tuple and publishes a non-persistent tuple asserting the end time of the event.

This producer is configured using a text file, which specifies the dataspace discovery multicast group and the four URIs that specify the Microphone-Position of each microphone.

5.6.3 iButton Reader Producer

The purpose of the iButton reader producer is to generate the iButton-Signed-In events that correspond to individual people signing in or signing out using their personal iButtons.

In terms of hardware, the producer consists of four individual iButton readers connected to a PC. An iButton reader is installed next to the microphone at each seating location. Similarly to the microphones, more than four iButton readers may be used if additional hardware is installed.

The producer software on the PC repeatedly polls the readers and detects when an iButton has been pushed into one of the readers and what the ID of the iButton is. When such a ‘sign-in’ occurs, the producer generates an iButton-Signed-In event, with the value of the id-of-ibutton-used property set to the iButton ID, and the has-location property set to the URI of the iButton-Reader-Position. This event is then published to the dataspace as a persistent tuple. The producer stores in memory the IDs of each of the iButtons currently signed in, so that when one of the already ‘signed-in’ iButtons is pushed into a reader again, it can determine that it is now a sign out event. When this occurs it deletes the previous tuple from the dataspace and then publishes a non-persistent tuple asserting the end time on the event.

This producer is configured using a text file, which specifies the dataspace discovery multicast group and the four URIs that specify the iButton-Reader-Position of each of the iButton readers.

5.7 Inference Engine and Triplestore

Both the inference engine and triplestore were implemented using the Jena 2.0 Semantic Web framework for Java [Jen04], developed by HP Labs. The framework provides an extensive range of features, which include:

- RDF parsing and serialisation.

- RDFS, DAML+OIL and OWL ontology handling.
- Support for persistent models in relational databases.
- Built in inference rules to allow automatic ontology-based entailments.
- A general purpose rule-based inference engine that supports user defined forward and backward chaining inference rules.

Its support for persistent models, OWL language based entailments and its general purpose rule engine are the features that made Jena 2.0 particularly suited to implementing the inference engine and triplestore. This section will give a technical description of these two components. To improve readability, a full description of the inference rules and logic used in the proof of concept implementation will be omitted from this section and discussed in chapter 6.

5.7.1 Overview of Relevant Jena Functionality

RDF graphs in Jena are called models. These act as a store for triples and may have triples added or removed. The triplestore is implemented as a Jena persistent model, held in a MySQL [MYS04] relational database. The particular model type used was a Jena OntModel, which is an ontology aware model, that when queried will return triples entailed from the ontologies, as well as triples that are explicitly part of the model. Jena provides an RDQL (RDF Data Query Language) [RDQ03] query interface for persistent models, which are accessed using JDBC (Java Database Connectivity).

The inference engine was implemented using the Jena general purpose rule engine, configured to run in forward chaining mode. The inference functionality is exposed as a model (called an InfModel) to which is bound a rule-based reasoner. Each rule consists of a list of body terms (premises) and a list of head terms (conclusions). Each term can be either a triple pattern or a call to an external piece of Java code called a 'builtin', which can be used to perform boolean tests or some other function.

For example, the rule given below is used to infer that if there is a Person-Present event and a Meeting-Taking-Place in that room, then that person should be added as a meeting participant. In addition to triple patterns, the rule also uses the builtins 'noValue', 'eventNotInMeeting' and 'print'. The complete rule set is given in Appendix F.

```
[addPersonToMeeting:
(?a rdf:type live:Person-Present),
(?a portal:has-location ?loc),
(?loc location:is-located-in ?room),
(?room rdf:type location:Meeting-Room),
(?meeting rdf:type portal:Meeting-Taking-Place),
(?meeting portal:has-location ?room),
(?meeting support:has-time-interval ?time),
noValue(?time support:ends-at-time-point),
eventNotInMeeting(?a),
(?a support:has-time-interval ?pptime),
noValue(?pptime support:ends-at-time-point),
(?a portal:has-main-agent ?person),
->
print("addPersonToMeeting has fired"),
(?meeting portal:has-sub-event ?a),
(?meeting portal:meeting-attendee person)
]
```

A specific rule fires when all the triple patterns in its body term match with triples already in the InfModel. When the body contains any builtins, all these must also return true before the rule will fire. When a rule fires, each of the triples defined by the triple patterns in the head terms are added to the InfModel. If the head contains any builtins, their Java code is run also. It is often the case that the new triples defined by a rule firing will in turn cause further rules to fire. This cascade of rule firing continues until no more rules can fire. Clearly care must be taken to avoid writing rules that will loop indefinitely. If triples are added directly to the InfModel using the Jena API, this can also trigger further rule firings. The forward chaining rule engine is implemented using the RETE algorithm [For82], which is optimised for such incremental changes.

5.7.2 Triplestore

The purpose of the triplestore was to provide additional knowledge that can be used as part of the inference process. It was also used by the display panel application to resolve URIs into human readable names.

A one-off initialisation of the triplestore was performed by populating it with the ontologies described in section 5.4, and some example instance knowledge required for the inference process. The instance knowledge to initialise the triplestore took the form of a number of hand authored triples specified using Notation-3 in a text file and some further triples automatically obtained by directly querying the AKT triplestore. Further details of this instance knowledge are discussed in section 6.5.

Since some of the initialisation knowledge was obtained from the AKT triplestore, it may seem more sensible to have used that triplestore directly as the system triplestore, and add the example instance knowledge to that triplestore. This would have been entirely possible, but it was chosen to use a Jena persistent model since it had much more support for generating OWL language entailments in response to queries. Furthermore, running a private instance of a triplestore made it quick and easy to assert and un-assert triples during development.

5.7.3 Inference Engine

The inference engine was an unusual component in that it acted simultaneously as both a producer and a consumer. In its consumer role, it joins the Equip4j dataspace for a collaboration session and subscribes to all the event types generated by the other producers, i.e. Joined-To-Session, iButton-Signed-In, and Microphone-Active. As the other producers publish tuples to the dataspace, the inference engine receives these tuples as soon as they are published and adds the triples they contain directly to its InfModel. Adding these triples may satisfy the conditions for the one or more rules to fire, which may in turn assert more triples, potentially causing further rules to fire.

The producer role is achieved as follows. Whenever a rule fires that infers a new event, in addition to the triples describing the event being added to the InfModel, they are also published to the dataspace (just as other producers do) using a special builtin created specifically for this purpose.

5.7.3.1 Archiving

At the end of a collaboration session, the InfModel contains a full description of all the session events (both captured and inferred). At this point a rule fires which calls a builtin to upload the triples to the triplestore in order to archive the session. As some of the triples in the InfModel will have originated from the triplestore, the builtin checks each triple to see if it is already in the triplestore. This avoids uploading duplicates. Once all the triples have been successfully uploaded to the triplestore, the builtin clears the InfModel so that it is ready for use when a new collaboration session begins.

5.7.3.2 Configuration

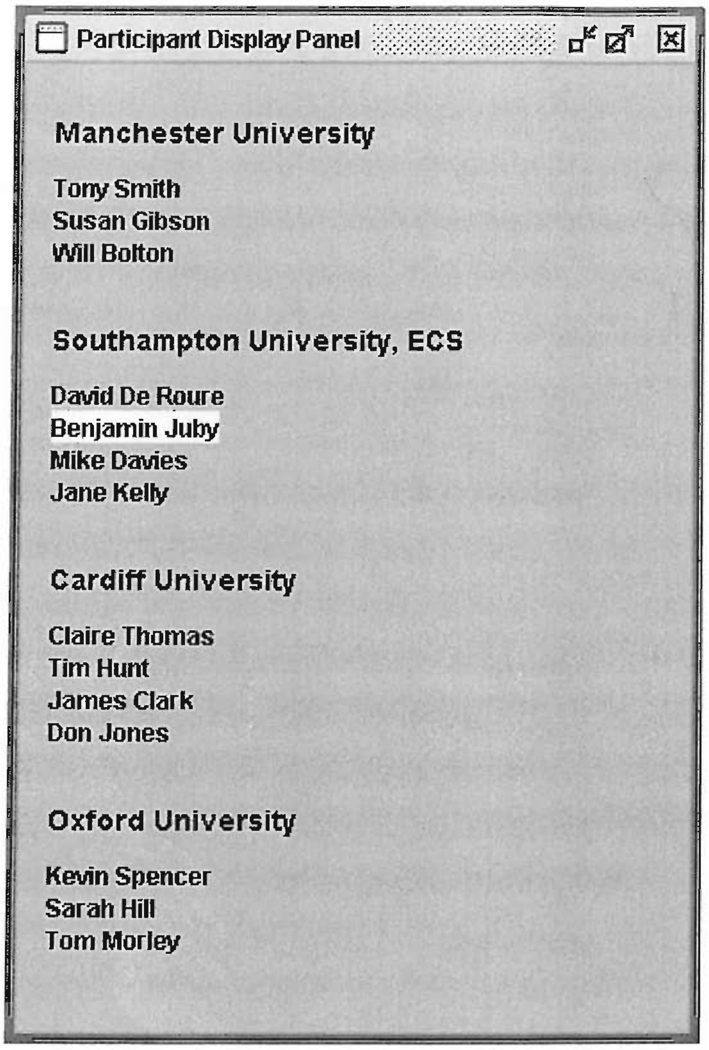
The bulk of the inference engine configuration consisted of the set of rules it used. These were hand authored. In order that it could access the triplestore, it also needed to be configured with the URL, username and password to access the MySQL database.

The inference engine did not use the EQUIP discovery mechanism used by the other components as it is intended that each instance the inference engine should be permanently bound to a given dataspace. Instead of using the discovery mechanism, the dataspace URL is simply passed to the inference engine when it is first run.

5.8 Consumers

The only pure consumer application implemented was the participant display panel. An instance of this consumer was intended to run at each site. It consisted of a single window designed to be displayed to participants at each site using the Access Grid projection screen. It was responsible for displaying the names of the individual sites currently joined to the session and the names of the people currently participating at that site. This information was updated in real-time to reflect changes in the session state, such as participants joining or leaving. Whenever a participant spoke, their name was highlighted in yellow to aid with speaker identification. Figure 5.7 shows a screenshot of the participant display panel, indicating that Benjamin Juby is currently speaking. Figure 5.8 shows a mock up of how the display panel would appear with the video windows in a running meeting.

The participant display panel functionality was achieved by the application subscribing to Joined-To-Session, Person-Present and Making-a-Verbal-Comment events. After some consideration, it was decided that the best way to handle the incoming triples was to again use another instance of the Jena generic rule engine. The alternative would have been to use the Jena API directly, which would have proved to be somewhat fiddly compared to using the rules engine. Unlike the inference engine, where rules were primarily used to infer higher level knowledge about the session, here the rules were used as a convenient way of matching on specific patterns of triples and invoking appropriate custom builtins to render the text in the display panel.


Participant Display Panel

Manchester University

Tony Smith
Susan Gibson
Will Bolton

Southampton University, ECS

David De Roure
Benjamin Juby
Mike Davies
Jane Kelly

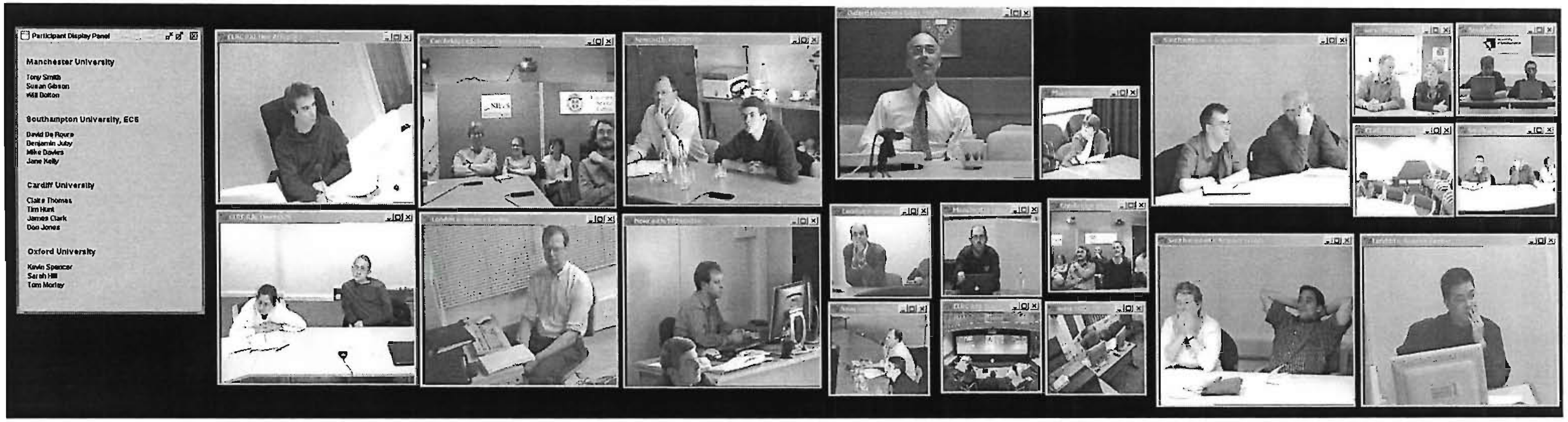
Cardiff University

Claire Thomas
Tim Hunt
James Clark
Don Jones

Oxford University

Kevin Spencer
Sarah Hill
Tom Morley

Figure 5.7, The Participant Display Panel


Participant Display Panel

Manchester University

Tony Smith
Susan Gibson
Will Dolton

Southampton University, ECS

David De Roure
Benjamin Juby
Mike Davies
Jane Kelly

Cardiff University

Claire Thomas
Tim Hunt
James Clark
Don Jones

Oxford University

Kevin Spencer
Sarah Hill
Tom Morley

Figure 5.8, The participant display panel in a running meeting

The display panel consumer had its own InfModel to which the triples were added as they arrived in the tuples. When the addition of new triples caused the terms in a rule body to match the triples in the InfModel, the rule fired, calling an appropriate builtin to update the text displayed.

There were two different types of rules. The first type queried the triplestore to obtain human readable text for resources (e.g. site names and participant names). The second type was responsible for displaying and updating the text in the display panel.

Like the inference engine, the bulk of the display panel application configuration consisted of the set of rules it used, and these were hand authored. In order that it may access the triplestore, it also needed to be configured with the URL, username and password to access the MySQL database. As it used the EQUIP discovery mechanism, it also was provided with the dataspace discovery multicast group.

5.9 Summary

This chapter described an implementation of the semantic annotation framework presented in chapter 4. The annotation functionality was based on the Access Grid scenario from chapter 3, and consisted of dynamically updated attendance lists combined with speaker identification. A novel technique was described for automated speaker identification that used existing Access Grid hardware.

An ontology was created to represent concepts such as events, time, location and people. A significant portion of this ontology was reused from existing ontologies, which reduced implementation effort and promoted potential interoperability. Three producers and a single consumer were created and these communicated by publishing sets of RDF triples to an EQUIP dataspace. EQUIP had the advantage of being specifically designed for supporting real-time collaboration and also had a useful discovery mechanism.

The inference engine and triplestore were provided by Jena, and a number of rules were created to describe the inference logic. In addition to the generic rules based inference, Jena's support for OWL entailments meant that some location inferences could be performed without the need for authoring extra rules. Reuse of instance data was also demonstrated by reusing participant name information from the AKT triplestore.

6 Details of the Inference Process

This chapter describes in detail the inference process used by the proof of concept system. It starts by giving a full description of each of the rules used by the inference engine. An explanation is then given of each Jena buitltin used as part of the inference process. These builtins allowed Java code to be called directly from within inference rules. Then a description is given of the bootstrapping knowledge that the inference process required and the chapter then finishes with a step-by-step walkthrough of an example collaboration session, showing how individual rules fire as a result of meeting room events.

6.1 Creation of Rule Set

The process that resulted in the creation of the rule set began with mental run-throughs of hypothetical collaboration sessions. In these run-throughs the different key sequences in which the events could be generated by the producers at the beginning and end of collaboration sessions were noted. After this, the inferences that could be made from these states were noted too.

This resulted in a set of logic which highlighted a number of key session states and the inferences that could be made from these. This logic was then used to help determine the rule set required, with the rules being one possible formal definition of this logic.

When creating the rules, the key design decision that had to be made was to either use a relatively small number of complex rules or to use a larger number of simpler rules. In the former case, there would be more tests in the rule bodies and more triples would be asserted by each rule, whereas in the latter case the rules would have fewer tests and would assert fewer triples, instead using cascading firing of rules where possible to make complex assertions.

Here the decision was to use multiple short rules where possible, as otherwise it would have resulted in some functionality being replicated between rules. Hence by avoiding this replication of functionality using multiple simpler rules made the rule set shorter (in terms of lines of code) and easier to modify during development, as changes to functionality could often be achieved by modifying a single rule, rather than multiple ones.

6.2 Classification of Rule Types

Overall there were three distinct types of rules used by the inference engine:

- The first type of rule are those whose head action is to query the triplestore for further triples and add these triples to the InfModel (see section 5.7.1), which may in turn allow further rules to fire. A new builtin was created for querying the triplestore in this way. An example of when such a rule may be used is when details of a new iButton are added to the InfModel as the result of an iButton-Signed-In event and the triplestore must be queried to determine who the iButton belongs to before further rules may fire.

- The second, and most common type of rule, are those which infer new triples from those triples already in the InfModel. An example of this type of rule is one that infers a Person-Present event from an iButton-Signed-In event and the iButton ownership information obtained from the triplestore.

- The third and final type only occurs once, and that is to archive to the triplestore at the end of a session.

As has already been mentioned, builtins were used as not all the inference engine functions could be achieved purely by using triple patterns in rules. In this implementation, the builtins were used for two key purposes:

- Performing logic tests in rule bodies that could not be expressed as simple triple patterns.
- Carrying out actions that rules alone could not perform (e.g. publishing inferred triples to the EQUIP dataspace and querying and uploading to the triplestore).

6.3 Inference Rules

This section gives a full description of each of the twenty inference rules used by the system, arranged using the classifications from the previous section. Each rule is named in bold, with the description following its name. The full text listing of the rules, as passed to the Jena rule parser are given in Appendix F. Each rule has been numbered in brackets and this corresponds to the numbering of the rules given in the appendix, enabling the reader to easily locate the listings for specific rules.

6.3.1 Rules That Query The Triplestore

Get Locations On Sign In (1)

This rule queries the triplestore the first time there is an iButton-Signed-In event at a particular iButton reader. This is done to determine the Seating-Position and Meeting-Room in which the iButton reader is located. Since is-located-in was declared to be a transitive property, a single query to the triplestore returns both the Seating-Position and Meeting-Room. This information is added to the reasoner's knowledgebase and is used in future inferences to, for example, determine the Seating-Position of a Person-Present event and to infer when participants are in the same Meeting-Room.

Get Locations On Microphone Active (2)

This rule is similar to 'Get Locations On Sign In' and queries the triplestore the first time there is a Microphone-Active event at a particular microphone. This is done to determine the Seating-Position and Meeting-Room in which the microphone is located. This information is used when Making-a-Verbal-Comment events are inferred to determine who made to comment (by using knowledge about which participant is in that Seating-Location) and to specify the Meeting-Room in which the verbal comment was made.

iButton ID To URI (3)

The purpose of this rule is to query the triplestore when there is an iButton-Signed-In event to resolve the ID of the iButton used to a URI that represents the specific iButton.

iButton To Person (4)

This rule fires after 'iButton ID To URI' has fired and queries the triplestore again to determine the person that specific iButton belongs to.

6.3.2 Rules That Assert New Triples

Create Person Present (5)

When there has been an iButton-Signed-In event and the relevant rules for querying the triplestore have fired, this rule then fires and is responsible for inferring a Person-Present event. It does this by mapping the iButton ID to the person who owns it and creating a Person-Present event for that person. As Person-Present events are one of the event types that the display panel application subscribes to, the rule also publishes the new inferred event to the EQUIP dataspace.

Create Single Meeting In One Room (6)

This rule fires when there are no other meetings in progress and a total of two Person-Present events in a single Meeting-Room. Given that there are two people in the Meeting-Room, it infers that there is a Meeting-Taking-Place in that Meeting-Room.

Create Meetings In Two Rooms (7)

If there are no other meetings in progress and there are a total of two Person-Present events, but in different Meeting-Rooms, then a Meeting-Taking-Place must be created for both Meeting-Rooms. This rule is responsible for inferring this. From the scope of the dataspace, it is implicit that both the participants are part of the same collaboration session and although there is only one participant present at each site, they are in a meeting with each other. Hence meetings are created even though there is only one participant at each Meeting-Room.

Add Person To Meeting (8)

This rule fires when there is a Meeting-Taking-Place at a specific Meeting-Room and there is a Person-Present event at that room that is not currently part of the meeting. This can either be due to the meeting being created after the Person-Present event (e.g. when Create Single Meeting In One Room fires) or when a person joins a meeting that is already in progress. This rule adds the Person-Present event to the Meeting-Taking-Place as a sub-event and also adds the person as a meeting-attendee.

Create Distributed Gathering (9)

This rule fires once there are two instances of Meeting-Taking-Place events. It infers that a Distributed-Gathering should be created now there are two meetings in session and that both those meetings should be added to the Distributed-Gathering as local events. Note that there can only ever be one Meeting-Taking-Place at a given Meeting-Room and that from the scope of the dataspace, it is implicit that both the meetings are part of the same collaboration session, hence the inference of a Distributed-Gathering is a valid one.

Create Additional Meeting (10)
Once at least one meeting is in progress, this rule fires to create a further new Meeting-Taking-Place whenever the first participant signs in at a new site. In a similar way to 'Create Meetings In Two Rooms', this rule infers a meeting at the new site even though there is only a single participant at that site, since this rule only ever fires after there is at least one other Meeting-Taking-Place. Note that this rule will only ever fire after an initial meeting or meetings have been created by either 'Create Single Meeting In One Room' or 'Create Meetings In Two Rooms'.

Add Meeting To Distributed Gathering (11)
When there is a Distributed-Gathering in session and a new Meeting-Taking-Place is created, this rule fires and adds the Meeting-Taking-Place to the Distributed-Gathering by asserting that it is a local event of the Distributed-Gathering.

Handle Sign Out (12)
When somebody signs out of a meeting using their iButton, this rule is responsible for asserting an end time on the Person-Present event that represented the person being present at the meeting. This shows that the person is no longer present at the meeting. As the display panel application subscribes to Person-Present events, this rule also publishes the end time to the dataspace.

End Meeting During Distributed Gathering (13)
This rule fires when there is a Distributed-Gathering in session and all the people have signed out of a meeting at a particular site. Since there are no more participants at that particular meeting, this rule asserts an end time on that Meeting-Taking-Place to indicate that it has now finished.

End Distributed Gathering (14)

This rule fires when all but one of the meetings in a Distributed-Gathering have ended. Since there is only one meeting still in session, the Distributed-Gathering must have ended, so this rule asserts an end time on the Distributed-Gathering to indicate this.

End Meeting After Distributed Gathering (15)

Since one meeting will still be in session (i.e. the last site with people still signed in) after a Distributed-Gathering has ended, this rule is responsible for ending this last meeting by asserting an end time on the Meeting-Taking-Place once enough people have signed out. Unlike 'End Meeting During Distributed Gathering', which ended the meeting once it had zero participants, this rule ends a meeting once it has one participant left. This is because as there are no other meetings in session (and hence participants) to collaborate with, so the final meeting must be over once it is down to its last participant.

End Meeting Before Distributed Gathering (16)

This rule handles the unusual case where a collaboration session has just a single meeting (i.e. at just one site) and a Distributed-Gathering has not yet formed (and may never form if other sites do not join) and then enough participants sign out of this meeting for it to end. Like 'End Meeting After Distributed Gathering', this rule will end this single meeting once it has just one participant left and it does this by asserting an end time on the Meeting-Taking-Place. Note that the only reason this rule is separate to 'End Meeting After Distributed Gathering' is that it is not possible in a single rule to specify the logic required (at least not without creating a specific builtin) to match on the distinct cases of either there being no Distributed-Gathering whatsoever, or there being a Distributed-Gathering, but that has now ended. This is why two rules were required to perform such similar tasks.

Create Verbal Comment In Meeting (17)

This rule is responsible for inferring a Making-a-Verbal-Comment event from Microphone-Active and Person-Present events. It does this by matching on the Person-Present event that is-located-in the same Seating-Position that the Microphone-Position is-located-in. It then matches on the has-main-agent property of the Person-Present event to determine who made the verbal comment. When this rule asserts a new Making-a-Verbal-Comment event, it also adds the event as a sub-event of the meeting the participant is in. As the display panel application subscribes to Making-a-Verbal-Comment events, this rule also publishes the new inferred event to the dataspace.

Create Verbal Comment Outside Meeting (18)

This rule is identical to 'Create Verbal Comment In Meeting', except that it only fires when a verbal comment is made when there is a Person-Present, but no Meeting-Taking-Place, and hence a Making-a-Verbal-Comment event is inferred, but is not specified as a sub-event of any meeting. There are only two scenarios where there can be a Person-Present event and no meeting. These are when the person is the very first to sign into a session or is the very last to sign out of a session. In both these scenarios, there will only be one participant present in the entire collaboration session, so it could be argued that there is no need to infer verbal comments in this situation as there is nobody else for the participant to speak to. While this is certainly true, the author feels that recording verbal comments in this situation is still potentially useful. For example, if the audio and video from a session were also being recorded, then a lone participant may wish to make a comment purely for the recording to perhaps serve as an introduction or a wrap up. In this situation having these comments annotated would clearly be useful.

Handle Microphone Active End (19)

The purpose of this rule is to assert an end time on a Making-a-Verbal-Comment event once the underlying Microphone-Active event from which it was inferred has ended. This shows that the verbal comment has finished being made. As the display panel application subscribes to Making-a-Verbal-Comment events, this rule also publishes the end time to the dataspace.

6.3.3 Rule To Archive The Session

Archive Session (20)

The purpose of this rule is to archive a collaboration session to the external triplestore once the session has ended. This rule determines a session is over once the very last participant signs out of the session. When this rule fires it calls a builtin which uploads the entire contents of the reasoner's knowledgebase to the triplestore and also clears the knowledgebase of triples, so it is ready for the next session.

6.4 Jena Builtins

The majority of the functionality of the Inference Engine was achieved through rules that had triple patterns in the rule bodies and asserted new triples in the rule heads. Unfortunately, not all the functionality described in the section 6.3 could be achieved purely by matching triple patterns in rule bodies and asserting new triples based on these patterns in rule heads. Fortunately Jena allows Java code to be called directly from within rules using builtins (see section 5.7.3 for more details). A number of builtins were created by the author to achieve specialist functions, and these were used along with a number of standard builtins that already came with the Jena distribution. These builtins are described here.

6.4.1 Standard Jena Builtins

This section describes how the standard predefined builtins that came as part of the Jena distribution were used.

noValue

This builtin takes two arguments: x and p, and returns true if there is no known triple (x, p, *) in the reasoner's knowledge base (where * represents a wildcard). It is used in a number of ways in this proof of concept application.

Firstly, it is used as a check before querying the triplestore to ensure that it has not already been queried for the same information, thus eliminating redundant queries.

Secondly, it is used to tell if an end time is asserted on an event. This is used to determine if an event is still active or not.

Thirdly, it is used as a way to prevent some rules firing twice because it can check if triples are present that have been asserted when the rule fired. This is needed because some rules such as 'Create Meetings In Two Rooms' that match on two events of the same type can fire on the same set of data twice, with the events in a different order.

notEqual

This builtin takes two resources as an argument and returns true when those resources are not equal. This is used in the rules that match on two different events of the same type to ensure that the rule does not just match on a single event twice.

print

The print builtin simply prints out text to the standard output. This text can either be the URI of a resource, a literal or any other string defined by the user. This proved to be extremely useful for debugging. The final set of the rules uses this builtin in each rule to show when each rule has fired by displaying an appropriate message.

6.4.2 New Builtins

This section describes the new builtins created by the author and how they were used. The discussion is partitioned into two sub-sections that handle respectively the builtins that performed logic tests and those builtins that performed other kinds of actions.

6.4.2.1 Logic Tests

noValue3

This builtin was identical to the Jena predefined noValue builtin, except that it took three, rather than two arguments. This meant that it could match on all three values of a triple, rather than just the subject and predicate. This was used to determine if a Distributed-Gathering had already formed, by checking if there were any resources of type Distributed-Gathering. Note that at the time of writing, the current version of Jena (version 2.1) now supports this with the standard noValue builtin.

noMeetingAtPhysLoc

This builtin takes the URI of a room location and returns true only if there is not currently a meeting in session at that location. What is meant by there being no meeting in session is that, within scope of current collaboration session, there has never been a meeting or there has been one but it has now ended. This is, for example, used in the rules that infer new meetings to check that there is not already a meeting in the room where they are about to infer a new meeting.

eventNotInMeeting

This builtin takes the URI of any event that can be a sub-event of a meeting and returns true only if that event is not currently a sub-event of any meeting (i.e. is not part of a meeting). This builtin is used in all rules that make inferences from Person-Present events (such as those that infer new meetings) and it ensures that rules only fire on Person-Present events that are not already sub-events of a meeting, since we would not wish to infer a new meeting from Person-Present events that are already sub-events of existing meetings.

eventNotInDistGath

This builtin takes the URI of any event that can be a local-event of a Distributed-Gathering and returns true if that event is not currently a local-event of any Distributed-Gathering. This builtin is used in 'Add Meeting To Distributed Gathering' to determine when a meeting is not currently part of a Distributed-Gathering.

participantsPresent

This builtin is used to determine if the number of participants present at a specific meeting is above or below some specified threshold. It takes three arguments, the first it the URI of the Meeting-Taking-Place to be tested, the second is the test to be performed, which is either "<=" (less than or equal to) or ">" (greater than), and the third is a number, which specifies the participant threshold. For example, the following call will return true only when the number of participants present in the meeting specified by `?meeting` is less than or equal to one:

```
participantsPresent(?meeting,"<=","1")
```

This builtin is used in the rules for ending meetings to determine when the number of participants in the meeting has reached the threshold for ending the meeting.

onlyOneMeetingInSession

This builtin takes no arguments and returns true only when there is currently one meeting in progress. This is used in the 'End Distributed Gathering' rule as part of the logic for determining when a Distributed-Gathering should be ended.

eventHasMostRecentEndTime

This builtin is used to determine if a particular event that has ended has the most recent end time out of a specified set of events. It takes a variable number of arguments. The first argument is the URI of a Time-Point to be tested, the second argument specifies the event type, which can be used to specify a Person-Present event or a Meeting-Taking-Place event. The third argument is only used when the event type is Person-Present and specifies a room location, which limits set of events to be tested. In this case, the builtin will only return true when there are no Person-Present events at that room with a more recent end time. When the event type is Meeting-Taking-Place, the builtin returns true when there are no Meeting-Taking-Place events in the reasoner's knowledgebase that have a more recent end time than the time being tested (irrespective of location).

When invoked for a Person-Present event, this builtin is used in all the rules that end meetings. Once the number of participants has reached the required threshold for the meeting to end (as tested by the participantsPresent builtin), then the eventHasMostRecentEndTime builtin is used to ensure that the rule matches on the most recent Participant-Present event to have ended, as it is the end time of this event that is taken as the end time for the meeting.

When invoked for a Meeting-Taking-Place event, this builtin is used in the rule 'End Distributed Gathering', and it ensures that the rule matches on the most recent meeting to have ended, as it is this end time that is taken as the end time for the Distributed-Gathering.

6.4.2.2 Actions

queryTriplestore

This builtin allows the external triplestore to be queried from within rules. It takes three arguments which specify a triple pattern. In the triple pattern, one or two of the arguments will be variables and the rest are resources or literals. The variables are treated like wildcards and it queries the triplestore for all triples that match the triple pattern. The triples returned by the query are added directly to the reasoner's knowledge base.

makeResource

The makeResource builtin takes a single variable as an argument, to which it binds an automatically generated unique URI. This is used whenever a new event is inferred, with the automatically generated URI used to represent the new resource.

publishToDataspace

The publishToDataspace builtin is used to publish inferred triples to the EQUIP dataspace as tuples or tuple events. It is a rather complex builtin that takes a variable number of arguments and is stateful between calls. It is called multiple times in a rule head to build up a tuple or tuple event before publishing it to the dataspace. Although it takes a variable number of arguments, the first argument is always a literal string, which specifies the type of operation to be performed.

The first call is always used to specify whether a tuple or tuple event is required. This operation type is specified by the string "TUPLE_TYPE" and the valid types are "TUPLE" (for the beginning of a meeting event) or "TUPLE_EVENT" (for the end of a meeting event). For example the following call specifies a tuple:

```
publishToDataspace("TUPLE_TYPE", "TUPLE")
```

The second call is used to specify the RDF type of the event that the tuple or tuple event represents (this is done to enable consumers to subscribe to the tuples or tuple events). For example, the following call specifies that the tuple contains a Person-Present event:

```
publishToDataspace("EVENT_TYPE", live:Person-Present)
```

The next calls are then used to add the triples that represent the meeting event to the tuple or tuple event. For example, the following two calls specify that the resource bound to `?pp_event` has an RDF type of Person-Present and a location of the resource bound to `?location`.

```
publishToDataspace("ADD_TRIPLE", ?pp_event, rdf:type, live:Person-
Present)
```

```
publishToDataspace("ADD_TRIPLE", ?pp_event, portal:has-location,
?location)
```

Finally, once the tuple or tuple event has been created through successive calls to the builtin, it is published with the following call:

```
publishToDataspace("PUBLISH")
```

Additionally, if the builtin has just been used to publish the end of a meeting event, the associated tuple that specifies the beginning of the event needs to be deleted. This is achieved by calling publishToDataspace with "DELETE" as the first argument and the URI of the event as the second argument. For example the following call will delete the tuple that that represented the beginning of the Person-Present event that is bound to `?pp_event`.

```
publishToDataspace("DELETE", ?pp_event)
```

getMostRecentTimePoint

This builtin takes three arguments. The first two are URIs of instances of support:Time-Point and the third is a variable to which the most recent of the two time points is bound. This buitltin is used in rules such as 'Create Distributed Gathering', where two events are used to infer an instance of a new single event (e.g. the presence of two meetings is used to infer that a single Distributed-Gathering is happening). In such a case then the new inferred event (e.g. Distributed-Gathering) needs to be given a time point which specifies when the event begun, and this time point should be the time point from the most recently created event from which it was inferred. E.g. a Distributed-Gathering starts as soon as the second meeting (i.e. most recent) is created, so should therefore take the beginning time point from that meeting, and not the earlier first meeting.

archiveSession

This builtin takes no arguments and when called uploads the entire contents of the reasoner's knowledgebase to the triplestore and also clears the knowledgebase of triples, so that it is ready for the next session. When uploading the triples, it checks each triple to see if the triplestore already contains that triple, since some of the triples in the knowledgebase will have originally come from the triplestore. This ensures that duplicate triples are not added to the triplestore.

6.5 Bootstrapping Knowledge

In addition to the inference rules, the system also required some explicit 'bootstrapping' knowledge to seed the inference process. This described the specific set of instances (e.g. people, meeting rooms etc.) that the system had knowledge of. All this knowledge was hand authored, apart from the human readable names for participants, which were automatically extracted from the CS AKTiveSpace triplestore. This bootstrapping knowledge was held in the triplestore component of the system and consisted of the following types of information:

- Information to map an iButton ID on to a specific person. For example, triples that specify the iButton that belongs to Benjamin Juby are shown below in Notation-3.

```
ecsinfo:person-03435 live:has-personal-identifier myibuttons:bpj00r .
myibuttons:bpj00r rdf:type live:iButton .
myibuttons:bpj00r live:has-ibutton-id "02000009EA6FD301" .
```

- Human readable names for participants, which were used in the display panel application. For example:

```
ecsinfo:person-03435 portal:full-name "Benjamin Juby" .
```

- Information about each Meeting-Room, specifying that the resource is of RDF type Meeting-Room and a human readable collaboration site name for use by the display panel application. For example, the following triples represent this information about the Southampton University, Electronics and Computer Science Access Grid room:

```
mylocs:sotonuni-B59-3241 rdf:type location:Meeting-Room .
mylocs:sotonuni-B59-3241 live:has-collaboration-site-name "Southampton ECS" .
```

- Information about each Seating-Position in each Meeting-Room, specifying its RDF type and which Meeting-Room it is located in. For example:

```
mylocs:sotonuni-B59-3241-seat1 rdf:type live:Seating-Position .
mylocs:sotonuni-B59-3241-seat1 location:is-located-in mylocs:sotonuni-B59-3241 .
```

- Information about each iButton-Reader-Position, specifying its RDF type and which Seating-Position it is located in. For example:

```
mylocs:sotonuni-B59-3241-seat1-reader1 rdf:type live:iButton-Reader-Position .
mylocs:sotonuni-B59-3241-seat1-reader1 location:is-located-in mylocs:sotonuni-B59-3241-seat1 .
```

- Information about each Microphone-Position, specifying its RDF type and which Seating-Position it is located in. For example:

```
mylocs:sotonuni-B59-3241-seat1-mic1 rdf:type live:Microphone-Position .
mylocs:sotonuni-B59-3241-seat1-mic1 location:is-located-in mylocs:sotonuni-B59-3241-seat1 .
```

6.6 Walkthrough of an Example Meeting

To demonstrate the inference process in action, this section gives a step-by-step walkthrough of a simple example fictional collaboration session. The example features three sites called A, B and C. Table 6.1 shows the events that occur in the session and the rules that fire as a consequence of those events. Table 6.2 then gives some actual examples of the triples that are generated at certain points in the meeting. A full description of the processes taking place is given in section 6.6.1 after the tables. In order to be understood, the tables first require some explanation.

Time is represented vertically on table 6.1, with time progressing down the page. Each basic (i.e. non-inferred) meeting room event is represented by an individual row in table 6.1. Note that the time interval between rows need not be equal. The room location of each event is indicated by which column it is in. The rightmost column then shows which inference rules fire, and in what order, as a result of the event. The following events are represented in the table 6.1:

- **join** – This represents the beginning of a Joined-To-Session event.
- **leave** – This represents the end of a Joined-To-Session event.

- **in** – This represents the beginning of an iButton-Signed-In event (i.e. the act of signing in). The number after the event shows the number of the seating position at which the sign-in occurred.
- **out** – This represents the end of an iButton-Signed-In event (i.e. the act of signing out). The number after the event shows the number of the seating position at which the sign-out occurred.
- **on** – This represents the beginning of a Microphone-Active event. The number after the event shows the number of the seating position at which the microphone is located.
- **off** – This represents the end of a Microphone-Active event. The number after the event shows the number of the seating position at which the microphone is located.

Any event or rule firing shown in table 6.1 written in bold, with a following number in superscript, has a corresponding numbered section in table 6.2. Each of these numbered sections in table 6.2 gives an example of the actual triples that are generated as a result of that event occurring or rule firing. Please note that this numbering has no relation to the numbering assigned to the individual rules when they were described in section 6.3.

Time	Site A	Site B	Site C	Firing Rules
↓		join		
			join	
	join			
	in 1			Get Locations On Sign In iButton ID To URI iButton To Person Create Person Present
	in 2[1]			**Get Locations On Sign In**[2] **iButton ID To URI**[3] **iButton To Person**[4] **Create Person Present**[5] **Create Single Meeting In One Room**[6] **Add Person To Meeting**[7]

			Add Person To Meeting
	in 1		Get Locations On Sign In iButton ID To URI iButton To Person Create Person Present Create Additional Meeting Add Person To Meeting **Create Distributed Gathering**[8]
	in 2		Get Locations On Sign In iButton ID To URI iButton To Person Create Person Present Add Person To Meeting
		in 1	Get Locations On Sign In iButton ID To URI iButton To Person Create Person Present Create Additional Meeting Add Person To Meeting Add Meeting To Distributed Gathering
on 2			Get Locations On Microphone Active Create Verbal Comment In Meeting
off 2			Handle Microphone Active End
on 2			Create Verbal Comment In Meeting
off 2			Handle Microphone Active End
out 1			Handle Sign Out
	out 1		Handle Sign Out
		out 1	Handle Sign Out End Meeting During Distributed Gathering
	out 2		Handle Sign Out End Meeting During Distributed Gathering End Distributed Gathering End Meeting After Distributed Gathering
out 2[9]			**Handle Sign Out**[10]

			Archive Session
		leave	
leave			
	leave		

Table 6.1, A timeline of an example collaboration session showing rule firings.

1. Triples added by the 'in2' event
myns:signinevent1 rdf:type live:iButton-Signed-In . myns:signinevent1 support:has-time-interval myns:timeinterval1 . myns:signinevent1 portal:has-location mylocs:sotonuni-B59-3241-seat1-reader1 . mylocs:sotonuni-B59-3241-seat1-reader1 rdf:type live:iButton-Reader-Position . myns:signinevent1 live:id-of-ibutton-used "02000009EA6FD301" . myns:timeinterval1 rdf:type support:Time-Interval . myns:timeinterval1 support:begins-at-time-point myns:timepoint1 . myns:timepoint1 support:year-of "2004" . myns:timepoint1 support:month-of "10" . myns:timepoint1 support:day-of "26" . myns:timepoint1 support:hour-of "21" . myns:timepoint1 support:minute-of "6" . myns:timepoint1 support:second-of "26" . myns:timepoint1 meeting:millisecond-of "209" .
2. Triples added by the 'Get Locations On Sign In' rule firing
mylocs:sotonuni-B59-3241-seat1-reader1 location:is-located-in mylocs:sotonuni-B59-3241-seat1 . mylocs:sotonuni-B59-3241-seat1 location:is-located-in mylocs:sotonuni-B59-3241 .
3. Triples added by the 'iButton ID To URI' rule firing
myibuttons:bpj00r live:has-ibutton-id "02000009EA6FD301" .
4. Triples added by the 'iButton To Person' rule firing
ecsinfo:person-03435 live:has-personal-identifier myibuttons:bpj00r .
5. Triples added by the 'Create Person Present' rule firing
mynamespace:ppevent1 rdf:type live:Person-Present . mynamespace:ppevent1 portal:has-location mylocations:sotonuni-B59-3241-seat1-reader1 . mynamespace:ppevent1 support:has-time-interval mynamespace:timeinterval2 . mynamespace:timeinterval2 rdf:type support:Time-Interval . mynamespace:timeinterval2 support:begins-at-time-point mynamespace:timepoint1 . mynamespace:ppevent1 portal:has-main-agent ecsinfo:person-03435 .
6. Triples added by the 'Create Single Meeting In One Room' rule firing
mynamespace:meeting1 rdf:type portal:Meeting-Taking-Place . mynamespace:meeting1 support:has-time-interval mynamespace:timeinterval3 .

mynamespace:timeinterval3 rdf:type support:Time-Interval . mynamespace:timeinterval3 support:begins-at-time-point mynamespace:timepoint1 . mynamespace:meeting1 portal:has-location mylocations:sotonuni-B59-3241 .
7. Triples added by the 'Add Person To Meeting' rule firing
mynamespace:meeting1 portal:has-sub-event mynamespace:ppevent1 . mynamespace:meeting1 portal:meeting-attendee ecsinfo:person-03435 .
8. Triples added by the 'Create Distributed Gathering' rule firing
mynamespace:distgath1 rdf:type meeting:Distributed-Gathering . mynamespace:distgath1 support:has-time-interval mynamespace:timeinterval4 . mynamespace:timeinterval4 rdf:type support:Time-Interval . mynamespace:timeinterval4 support:begins-at-time-point mynamespace:timepoint2 . mynamespace:distgath1 meeting:has-local-event mynamespace:meeting1 . mynamespace:distgath1 meeting:has-local-event mynamespace:meeting1 .
9. Triples added by the 'out 2' event
mynamespace:signinevent1 support:ends-at-time-point mynamespace:timepoint3 . mynamespace:timepoint3 support:year-of "2004" . mynamespace:timepoint3 support:month-of "10" . mynamespace:timepoint3 support:day-of "26" . mynamespace:timepoint3 support:hour-of "21" . mynamespace:timepoint3 support:minute-of "47" . mynamespace:timepoint3 support:second-of "32" . mynamespace:timepoint3 meeting:millisecond-of "77" .
10. Triples added by the 'Handle Sign Out' rule firing
mynamespace:timeinterval2 support:ends-at-time-point mynamespace:timepoint3

Table 6.2, Examples of triples generated at specific points in the meeting.

6.6.1 Full Description of the Processes Taking Place

The session starts by each site asserting a Joined-To-Session begin event. This does not trigger any rules, but does enable the display panel application to show that sites are joined to the collaboration session, even if there are no participants present yet.

The first sign-in is at site A, and this triggers 'Get Locations On Sign In' which queries the triplestore for the Seating-Position and Meeting-Room that the iButton-Reader-Position is located in. This information is used in future inferences. 'iButton ID To URI' also fires and this queries the triplestore, to resolve the iButton ID to the iButton it belongs to. The presence of this new iButton in the knowledgebase then triggers 'iButton To Person' which queries the triplestore for the person who owns that iButton. After this query, the presence of this new person then triggers 'Create Person Present', which asserts a Person-Present event. This pattern of rule firings occurs whenever a person signs-in.

The second sign-in is at site A also, and after the standard sign-in rule firings, 'Create Single Meeting In One Room' then fires as a result of there being two Person-Present events in the same Meeting-Room (the location information asserted by 'Get Locations On Sign In' contributed to this inference). This rule asserts a Meeting-Taking-Place at site A. As both Person-Present events at site A are not yet sub-events of the Meeting-Taking-Place, 'Add Person To Meeting' fires twice to add both Person-Present events to the meeting.

The next sign-in is at site B, and after the standard sign-in rule firings, 'Create Additional Meeting' fires to create a new meeting at site B and then 'Add Person To Meeting' fires to add the person at site B to the newly created meeting. As there are now two meetings, 'Create Distributed Gathering' fires to create a Distributed-Gathering containing those meetings.

The next sign-in is at site B also, and after the standard sign-in rule firings, 'Add Person To Meeting' fires to add the new person to the meeting already taking place at site B.

The next sign in is at site C, and after the standard sign-in rule firings, 'Create Additional Meeting' fires to create a new meeting at site C, then 'Add Person To Meeting' adds that person to the new meeting. 'Add Meeting To Distributed Gathering' then fires to add the newly created meeting to the Distributed-Gathering that is already taking place between site A and site B.

There is then a Microphone-Active begin event at seating position #2 at site A. As this is the first Microphone-Active event at that particular microphone, 'Get Locations On Microphone Active' fires which queries the triplestore for the Seating-Position and Meeting-Room that the Microphone-Position is located in. 'Create Verbal Comment In Meeting' then fires, which infers from the location information asserted by 'Get Locations On Microphone Active' and 'Get Locations On Sign In' that the person sitting at seating position #2 has started Making-a-Verbal-Comment.

The Microphone-Active event then ends, causing ‘Handle Microphone Active End’ to fire. The sole purpose of this rule is to publish the end time on the Making-a-Verbal-Comment to the dataspace for the benefit of consumers.

There is then another Microphone-Active begin event at seating position #2, and as this event has occurred at that position before, ‘Get Locations On Microphone Active’ does not fire, as the triplestore has already been queried for the Seating-Position that the Microphone-Position is located in.

The first sign-out event is at site A, and this simply causes ‘Handle Sign Out’, which asserts an end time on the associated Person-Present event. The same happens for the next sign out at site B.

There is then another sign-out at site C, which causes ‘Handle Sign Out’ to fire. As there are no longer any participants in the meeting at site C, ‘End Meeting During Distributed Gathering’ fires to end this meeting.

The next sign-out is at site B, which causes ‘Handle Sign Out’ to fire. As there are no longer any participants in the meeting at site B, ‘End Meeting During Distributed Gathering’ fires to end this meeting. Furthermore as there is now only a single meeting at site A, ‘End Distributed Gathering’ fires to end the Distributed-Gathering. Since there is no longer a Distributed-Gathering in session, the criteria for ending a meeting now is that it should be ended once there is only one participant left (compared to zero participants left when there is a distributed gathering). As the meeting at site A has only one participant left, ‘End Meeting After Distributed Gathering’ fires to end this final meeting.

The final sign-out is at site A, which causes ‘Handle Sign Out’ to fire. Since there are now no participants in the session, ‘Archive Session’ fires.

6.7 Summary

This chapter has provided a detailed description of the rules used by the inference engine in the proof of concept implementation of the framework from chapter 4. The most common rule type were those that inferred new triples from those already asserted, but rules were also required to query and upload to the triplestore. The implemented Jena builtins were also discussed, as these were required to carry out functions that pure rules were not capable of. These functions fell into the categories of either logic tests or actions such as querying the triplestore or publishing to the dataspace.

There now remains two more chapters; the next chapter provides a discussion based evaluation of the framework and implementation and the final chapter presents the overall conclusions for the thesis.

7 Evaluation

This chapter presents a discussion-based qualitative analysis of the system framework and implementation. It starts with a discussion on the performance and then presents an evaluation of the system. The performance discussion helps justify the building of the prototype system as it uncovers some real-time performance features that could not be accurately predicted without implementing a real system. The evaluation then focuses on the semantic features, as this is where the majority of the novel work has been. The criteria used to evaluate the systems in the literature review, will also be applied here. The tools and technologies used to perform the implementation are also evaluated.

It was decided not to perform a user-based evaluation of the system, as this would have mainly evaluated the functionality that was passed on to session participants. While such an evaluation would be useful for evaluating the human factors, the key contributions of the implementation were infrastructure based. The system demonstrated a general purpose architecture that was capable of using inference in real-rime to combine facts obtained from multiple sources of knowledge as would be found in the Semantic Web. This key contribution would not have been evaluated by a user trial.

7.1 Performance

The speed performance of the implemented system was examined on a qualitative level. Initially the system was found to be extremely slow. Specifically, responses to queries to the triplestore (i.e. the Jena persistent model) were found to take in the order of a minute. Clearly this would not have been practical for a system intended to respond in near real-time. The reason for the slow response time was found to be due to the ontology level entailments being computed by the persistent model on-the-fly at query time.

These entailments were the transitive closure of rdf:type and also the properties declared as owl:TransitiveProperty. The transitive closure of rdf:type meant that for each query for the type of a class, the Jena model would not only return the type explicitly declared in the ontology, but would also work out all the other superclasses the class was implicitly an instance of.

The transitive property declaration on the location:is-located-in property meant that every time the persistent model was queried to find out which instance of location:Abstract-Space another instance of Abstract-Space was located in, it would also compute and return all the other instances of Abstract-Space higher up the location hierarchy too.

Both these types of entailments could potentially be useful, although in the implemented system, only the owl:TransitiveProperty entailments were harnessed, as these reduced the amount of location information that had to be explicitly added to the triplestore and also resulted in fewer queries to the triplestore. The rdf:type entailments could have been used to allow more generic rules to have been written that would match on instances of classes of several different types (providing they shared a common superclass), for example allowing rules to fire on any instance of location:Room, rather than the more specific location:Meeting-Room.

In order to speed up the query response time, the system persistent model was modified to not work out any entailments at query time, but instead to pre-compute the transitive property entailments when the triplestore was initialised. This was achieved by making a query for all triples that had an location:is-located-in property. This query returned all the entailed triples as well as the explicitly specified ones. These returned triples were then placed back into the triplestore.

After this single modification, the performance of the system improved dramatically. In terms of human perception, queries to the triplestore were performed near instantaneously and the display panel application performed without any perceivable lag, meaning that the rule-based inferences were being performed in near real-time, as was originally intended.

7.2 Semantic Aspects

The system shall be evaluated against the criteria taken directly from the motivational discussion in chapter 3. These criteria are inference, interoperability, reuse, extensibility and indexing, since these are arguably the key value-added features the semantic approach provides over non-semantic approaches.

7.2.1 Inference

Clearly inference was a central feature of the system. It was demonstrated on two different levels, namely generic rules based inference and language based entailments. The approach was able to combine knowledge from multiple sources and then assert facts that were otherwise only implicit from the input provided by the producers and the triplestore. Some of these asserted facts were then in turn be combined using inference again to assert further facts. This meant that the information generated by the consumers or held in the triplestore could be very basic and therefore simple to create. Yet despite this information being basic, meaningful and relatively complex functionality was built up through using inference.

For example, apart from the tuple space discovery address, the only configuration iButton readers required was a single URI. From this single URI and the ID of an iButton pushed into the reader, rules based inference was able to take these facts and combine them with another fact from the external triplestore that stated who owned that iButton. Each of these facts in isolation were very simple to generate, but had only very limited meaning. However, the inference process was able to use these simple facts to make the meaningful assertion that there was a specific, identified person located at that iButton reader.

From this inference, further meaningful inferences could be made. For example, it then allowed inferences to be made about when that person was speaking. As described in section 5.3.1, facts about the microphone being active, the seating position the microphone was located in, the seating position the iButton reader was located in were combined with the already inferred fact about the specific person being located at that iButton reader. These first three facts had only very limited meaning on their own, and yet though inference, these basic facts could be combined with another inferred fact to make a meaningful assertion about a specific person speaking.

Similarly, the inferred facts about participants being present were combined with facts about their seating positions and the relationship of those seating positions to a specific meeting room. This was used to determine when the number of participants was above a certain threshold and therefore make the inference about there being a meeting taking place between those people in that meeting room. Furthermore, transitive property entailments from the OWL language meant that the indirect relationship about the iButton readers being located in a meeting room (i.e. only specified in the bootstrapping knowledge through their relationship to seating positions) appeared as an explicit relationship when querying the triplestore. This resulted in fewer queries to the triplestore, as instead of querying for the iButton reader's seating position and then querying for the room which that seating position was located in, a single direct query could instead be made for the iButton reader's meeting room. This also simplified the inference rules, since instead of having to specify extra body terms to match on an indirect relationship, they could instead match on the direct relationship between an iButton reader and its meeting room.

Therefore it is possible to see that through using inference, meaningful functionality has been built up from very simple facts that are straightforward for devices to generate or to specify in an external triplestore. Even though each of the basic facts had very limited meaning when examined in isolation, combining them with inference meant that meaningful facts were obtained.

Looking specifically at the bootstrapping knowledge in the triplestore, using inference had clear advantages. It removed the need to explicitly specify all the relationships between all resource instances. This enabled the bootstrapping knowledge to potentially be more general purpose, as it did not need to assert all the specific relationships that were used by the implemented system. For example the triplestore only needed to specify the location of each meeting room device relative to a seating location, but the system was still able to use this information to make inferences about devices being located next to each other or in the same meeting room. Furthermore, when moving a device to a different location, updating the triplestore knowledge to reflect this would be very simple as only a single relationship would have to be altered.

As the Semantic Web becomes more distributed, its possible to envisage moving beyond a single triplestore to provide bootstrapping knowledge. With multiple distributed sources of bootstrapping knowledge, it would likely be the case that publishers of this information may not know all the explicit relationships between system resources, thus the ability to infer these relationships is an attractive feature.

Furthermore, though enabling fewer relationships to be specified in fewer components, it could potentially reduce the chances of contradictory relationships being accidentally specified. Inference therefore also has the potential to simplify maintaining consistency between multiple knowledge sources.

7.2.2 Interoperability and Reuse

The system provided excellent interoperability due to common ontologies being shared by the different components. Interoperability with an independent external component was demonstrated by the system using knowledge about the names of participants held in the CS AKTive Space triplestore. This knowledge was automatically obtained from this triplestore and was added to the other bootstrapping knowledge used by the system. The knowledge pulled out of this triplestore did not need any modification and integrated seamlessly with the other bootstrapping knowledge despite the fact that the CS AKTive Space triplestore was developed independently.

Interoperability between the diverse components within the system was also demonstrated. The system consisted of varied producers, consumers, an inference engine and a triplestore. Through the use of RDF and a shared ontology, the information that originated from each component was completely interoperable with the other components in its native form. For example, the inference engine needed to draw no distinction between facts asserted by producers and facts obtained from the triplestore, despite these components having very different roles in the system.

The use of formally defined ontologies would also aid other external systems in interoperating with the proof of concept system, giving a clear specification of the vocabulary that the external system would have to use if it, say, wanted to reuse the information held in the triplestore. In fact because the proof of concept system reused the ontology from the CoAKTinG project, it meant that the CoAKTinG meeting replay tool would be able to use the annotations generated by the system.

Furthermore, had it been required to integrate the system with a component that used a different external ontology, this could have been achieved quite simply and rather elegantly by creating a new mapping ontology to map terms in the external ontology to terms in the existing ontologies.

The system demonstrated reuse on two different levels, namely ontology level reuse and instance level reuse. At the ontology level, it came from reusing the AKT, CoAKTinG and Signage location ontologies, which not only significantly reduced the amount effort to create an ontology for the implementation, but also potentially allowed straightforward interoperability with existing tools.

At the instance level, it came from reusing the name information held in the CS AKTive Space triplestore, which reduced the amount of effort required when specifying knowledge required by the system. Admittedly the instance level reuse was on a fairly small scale as it was just restricted to people's names. However, a more complex application could potentially reuse more information about people, or other resources such as projects.

7.2.3 Extensibility

The architecture was well suited to being extended. The main ways in which it is possible to envisage the system being extended are through the addition of new producers and event types, or the addition of new consumers for either displaying new events or existing events in a different form.

By using inference to enable the bootstrapping knowledge to record only a small number of explicit relationships between resources, it means that the addition of new components such as producers would require only minimal changes to this knowledge, since their relationship to the other components can be inferred, rather than having to be explicitly stated. Similarly, the existing producers would not have to be modified, as they do not need to have any knowledge of the other components in order to generate their simple, isolated facts. The pub/sub model of a tuplespace also means that there is no need for producers and consumers to be aware of the existence of each other in order to communicate.

Furthermore, the ability of inference to enable the knowledge within the system to be distributed (see section 7.2.1) also means that new sources of bootstrapping knowledge could be easily 'bolted on' without having to explicitly integrate it with the existing bootstrapping knowledge.

The standard OWL extensibility mechanism that allows anybody to import an existing ontology and add new terms is well suited to allowing people to define new event types. New rules may also easily be added to the rule set of the inference engine to handle or generate new event types. An open issue, however, is who performs extensions or modifications to the rule set. While anybody is free to add producers, consumers or extend the ontologies, the rules reside on a centralised inference engine and may only be modified by those people with administrative rights for the engine. This is a limitation to truly open extensibility, as many extensions to the system would require modification of the rules and clearly only trusted people may be allowed to modify the rule set.

7.2.4 Indexing

As stated in chapter 5, a feature for replaying indexed video from meetings was not implemented since this feature had already been implemented by the CoAKTinG meeting replay tool, and is currently being actively developed by the author for the Memetic project. The ability of the implementation presented in this thesis to be able to automatically generate annotations (which can then be used as indices) was a vital feature, since the existing CoAKTinG tool relied heavily on labour intensive manual annotation.

The annotations generated in the implementation were ideal for use as indices, since they were timestamped with both start and end times that identified the precise portions of the meeting where the events occurred. The use of UTC timestamps also meant that the system would work for meetings distributed across timezones.

The chosen annotation types were also well suited to providing meaningful indices to meetings. For example, participant tracking would allow a replay start at the point when a particular participant joined the meeting. Alternatively, if a participant had to leave the meeting mid session, they could later easily locate that point in a meeting recording and start the replay from there to catch up on what they missed. Similarly, speaker identification information presented as a timeline makes it straightforward to jump to sections where a specific participant was the primary speaker. Indeed during initial discussions, end users on the Memetic project have stated that they would like to use annotations about who was in a meeting and who was speaking as indices for recorded meetings.

Furthermore, by using the formally defined ontology for the annotations, the index can be used by computers as well as people. This could potentially be used to automatically edit meetings or re-purpose material. For example, long meetings sometimes have intentional coffee breaks etc. where the meeting stops for the participants to leave and then later resume the meeting. A video recording component may be left running during these breaks. Using annotations about when the meeting was in session, an automated editing component could remove these sections of the video from the recording.

Its possible to envisage automated re-purposing being of use in domains like television news, where the production team may wish to locate specific sections of meetings between politicians etc. for inclusion in broadcast news items.

7.3 Criteria from Literature Review

Section 2.2 of the literature review reviewed a number of existing systems according to a set of criteria. Of those criteria, two have not been discussed yet in this evaluation and it is useful for them to be applied to the proof of concept system and be discussed here. These criteria are support for live processing and degree of automation.

7.3.1 Support for Live Processing

The system had excellent support for live processing, which meant that the value added by the system could be taken advantage of during live collaboration sessions, as well as after sessions. The qualitative testing has shown that the tuple space, inference engine and triplestore each perform in near real-time, the net result of which is that the display panel application could display useful information to session participants in near real-time.

One area in which live processing has not been explored is for building up an index of the session on-the-fly in a replay client. This could be used to replay earlier sections of a collaboration session still in progress to help late joining participants or refresh group memory after a digression, in a similar way to the Where Were We system [Min93], which used hand authored index marks.

In order to achieve on-the-fly indexing, a replay client would join the tuple space and subscribe to all the different events it requires to build up an index on-the-fly. In this case, it would be preferable not to delete events from the tuple space once they have ended, as this would allow a late joining replay client to obtain a full session history.

7.3.2 Degree of Automation

The system supported a high degree of automation, especially from the perspective of session participants, leaving them free to focus on the collaboration and not have to make any significant effort to author annotations. The only additional task that the system required participants to do was to use a personal iButton to sign into or sign out of the session. It is easy to envisage other biometric participant identification and location tracking techniques such as face recognition that would no longer make it necessary for participants to carry iButtons and explicitly sign in or out.

The system operator (e.g. the Access Grid node operator) also had to perform only very simple tasks to initialise the system and then later teardown a session. It is possible to envisage these functions being integrated directly into the software that handles the videoconferencing session meaning that they could all be done automatically when the operator starts up and tears down the videoconferencing session.

The system did require some hand initialisation of its bootstrapping knowledge, for example to specify information about seating, microphone and iButton reader positions and also to specify the iButton that belonged to each person. In a deployed system this knowledge would also have to be maintained, for example being updated when new personnel joined or when meeting room layout was changed. Some of this knowledge could be obtained from existing sources, as was demonstrated with the participant names taken from the CS AKTive Space triplestore, but it is unlikely that all the bootstrapping knowledge could be obtained from existing sources. For example information about a meeting room layout is not something normally specified in existing information sources.

7.4 Tools and Technologies

This section discusses the tools and technologies used to implement the proof of concept system and discusses any strengths or weaknesses exposed in the implementation.

7.4.1 RDF(S) and OWL

Using RDF(S) and OWL proved to be a good choice. One of the key benefits provided over other knowledge representation formats was integration with the Semantic Web. This provided massive potential for reuse of ontology and instance data, and this reuse of ontologies and instance data was demonstrated in the proof of concept system.

Only a small subset of the features of OWL were required by the system, nor did the implementation expose any features that were missing from RDF(S) or OWL. The extensibility mechanism of OWL was seen to work well, allowing multiple existing ontologies to be easily imported into the live collaboration ontology by just specifying the URIs, and also allowing classes to be extended through the RDFS inheritance mechanism. The use of owl:equivalentClass was also demonstrated to successfully integrate the Signage location ontology and the AKT Portal ontology. Additionally, the use of owl:transitiveProperty was used to reduce the amount of bootstrapping knowledge that needed to be hand authored, and also reduce the number of triplestore queries.

7.4.2 Ontologies

In addition to the creation of a new ontology, the implementation saw the reuse of the AKT Support, AKT Portal, CoAKTinG meeting and Signage location ontologies.

One area of the live collaboration ontology that warrants further discussion was the chosen representation for the locations of iButton readers and microphones. This representation was based on an extension of the Signage location ontology. The Signage ontology was created in such a way that there was some degree of duality between the concept of location and the physical object that defined that location. For example a building can either be thought of as physical object consisting of bricks and mortar, or as some form or enclosed space. While this duality is intuitive for things such as buildings or rooms, it is not intuitive for things such as iButton readers and microphones. This means that the classes iButton-Reader-Position and Microphone-Position, proved to be counter intuitive representations.

It would have been more intuitive to use classes that represented the physical devices, rather than their positions. A more sensible representation may have been to define a class called Device and subclass it into the classes iButton-Reader and Microphone.

These would then represent the actual physical devices, rather than their location. This representation could then be tied into the existing Signage ontology by giving the Device class an is-located-in property with a range of Abstract-Space. Furthermore the Event class could be extended with a new property such as device-of-origin, to record the device that generated the event.

One particular feature from that AKT support ontology that was not ideal was its representation of time. It was a highly verbose format that (including the CoAKTinG milliseconds extension) used seven instances of owl:DatatypeProperty to specify a single point in time. The main effect of this was that it led to the inference rules containing a fairly large number of terms when having to match against time points or assert new ones. A better representation would have been a single numeric timestamp to represent a point in time. It was chosen to use the AKT representation of time as it would potentially allow easier interoperability with other AKT tools, and in particular the CoAKTinG meeting replay tool.

7.4.3 Jena

The Jena 2 framework has been shown to be a powerful framework with many useful features. In particular, its support for rules based inference, OWL entailments and remote queries to persistent models were central to the proof of concept system.

The benefit of the rules based approach to inference was that it allowed logic to be simply and compactly specified that would otherwise have been very awkward to specify using the Jena API directly. Furthermore, the RETE algorithm used by the rules engine was ideal as it was optimised for precisely the kind of incremental updates to the knowledgebase that the proof of concept system used.

Conversely, the main drawback of the rules based approach was that there were some functions that it was very awkward or impossible to implement using rules alone, such as determining how many participants were present in a meeting or which time point out of a pair was the most recent. However, Jena's support for builtins in rules that could call regular Java code meant that these could be used whenever pure rules could not. Builtins were an especially powerful feature, that not only allowed complex logic functions to be performed, but also allow calls to external code, which could be used to perform tasks such as publishing EQUIP tuples or querying the triplestore. Therefore the rules engine in Jena allowed a hybrid approach in which rules could be seamlessly integrated with calls to Java code, giving the 'best of both worlds'.

The support for language based entailments was a useful feature, but its performance was too slow for these to be computed at query time and a workaround had to be done to pre-compute these. Cleary a useful addition to Jena's functionality would be an inbuilt facility for pre-computing entailments. When the entailments were pre-computed, the persistent model appeared to perform well and responded to queries in near real-time.

7.4.4 EQUIP

Equip4j (and a tuple space model in general) proved to be a good choice for implementing the event communications infrastructure. Its speed performance appeared to be very good and did not introduce any appreciable delay in the system. Using tuples meant that each event type could be directly exposed as a separate tuple field, allowing subscriptions to be straightforward. Furthermore the pub/sub model of a tuple space allowed the knowledge producers to be loosely coupled with the knowledge consumers, which would allow easy addition of new producers or consumers.

The tuple persistence mechanism of EQUIP was also a useful feature, as it allowed late joining consumers to retrieve the current meeting state. If inactive events were not deleted, it could also be used to allow a late joining consumer to retrieve the entire meeting history, which could be used by a replay client (as discussed in section 7.3.1).

The multicast based discovery mechanism was another handy feature and further promoted the loose coupled and dynamic nature of the system. One drawback was that it relied on local multicast being available, which is not a feature present on all networks. However, if multicast wasn't available, a local unicast only dataspace could have been used instead to perform a similar function.

Clearly as EQUIP was not originally designed for use in Semantic Web applications it is not ontologically aware and treats the data it carries as opaque values. One potential extension to EQUIP could be to extend its subscription mechanism to understand the RDF class hierarchy, allowing a subscription to a single event type to also automatically subscribe to all subclasses of that event also.

7.5 Other Issues

This section discusses a number of open issues that the implementation exposed. In particular these issues were error handling and Quality of Service.

7.5.1 Error Handling

One of the drawbacks of the approach taken during the implementation was that the final system was intolerant to certain kinds of errors. Specifically, it was unable to elegantly handle the cases when a query to the triplestore returned no results, or returned information that was incorrect. Assuming that the producers and inference rules were correctly written, the information in the triplestore was the weak link. The system had a requirement that all external knowledge it needed was held in the triplestore, and that this knowledge contained no errors. In a prototype system, this requirement was not unrealistic, but in a deployment situation it may not be realistic to expect that a triplestore would not contain any missing information or incorrect knowledge.

Missing information was a problem, as it could cause the inference process to halt, since the required information would not be present to infer other knowledge from. Incorrect information was also a problem, as it led to false inferences.

In the case of incorrect information, it would be difficult for an application to automatically determine that the information was incorrect, and even harder to automatically correct the information. An approach that could help identify incorrect information would be to query multiple triplestores. If the triplestores were not initialised from the same sources, then it may be possible to identify incorrect knowledge by looking for contradictions between the knowledge obtained from the different stores.

Missing information was straightforward to detect during a collaboration session, but like incorrect information, would be difficult to automatically correct. In the implementation, some basic functionality was created to handle some types of missing information. For example, when a query to resolve an iButton to the person who owned it returned no results, then a new instance of a person was automatically generated.

This meant that the inference process could continue, even if there was no other information about that person, such as their name.

It would also have been possible to implement further functionality that flagged such an occurrence to the node operator or to the participant concerned. This would then allow them to input further information about that new person instance, such as their name. If that person already had an entry in the triplestore, then it could be manually declared as owl:sameAs the automatically created instance.

This approach of creating a new resource when a query returned no results was not suitable for all query types in the implementation. For example, it was not suitable for location queries, since generating a new instance of Meeting-Room or Seating-Position, would not have helped with inferences about people or devices being located in the same seating positions or meeting rooms.

7.5.2 Quality of Service

The system framework assumed that the underlying network provided only best effort Quality of Service (QoS). While this fitted in well with the current Internet, its was far from ideal as it is preferable in a live session that each event has a defined time by which it must arrive. This can't be achieved with best effort QoS, unless late data is dropped, which is not acceptable in this framework. Initially it may appear that incorporating an existing QoS framework such as IntServ [Bla98] or DiffServ [Bra94] would provide a straightforward solution. However, different event types require a different QoS and this is something that existing QoS frameworks do not provide. For example real-time speaker identification data has fairly tight synchronisation requirements and should not be delayed for more than, say, a few hundred milliseconds, whereas participant sign-in data has much looser requirements where it could be delayed by several seconds and still be of use. It is difficult to see how these mixed requirements could be achieved using existing QoS frameworks.

7.6 Summary

This penultimate chapter has presented a qualitative discussion-based evaluation of the semantic annotation framework and implementation. This evaluation can be summarised as follows:

- The time performance of the implementation was qualitatively evaluated as near real-time, although the entailment features of Jena were not found to work in real-time. A built in Jena feature for pre-computing entailments would be useful. Overall it was shown that real-time inferencing during live collaboration was practical.

- The Semantic Web based approach provided excellent features for inference, interoperability, reuse, extensibility and indexing. A significant portion of the system ontology was reused from existing applications, which reduced implementation effort and also allowed potential easy interoperability with existing tools that used those ontologies.

- The rules based approach to the inference was a convenient format for specifying complex inference logic, but was limited in the types of logic it could specify. These limitations were overcome by invoking Java code directly from rules, which meant that compact, but powerful rules could be written.

- The implementation used a small subset of OWL and RDF(S) and did not expose any shortcomings in the languages. OWL's support for transitive properties meant that automatic entailments could be used to reduce the amount of system bootstrapping knowledge required and also resulted in fewer queries to the triplestore.

- Overall the existing ontologies used appeared to be good basis for the live collaboration ontology. The chosen location based representation of meeting room devices proved to be counter intuitive and an alternative representation was discussed. The representation of time taken from the AKT support ontology also was found to be verbose and awkward to work with when writing inference rules.

- A weakness of the framework was that it required the information in the triplestore to be complete and correct, which may not be the case in real-world situations. Missing information could cause the inference process to halt and incorrect information could lead to false inferences.

- A tuple space model was shown to be an appropriate basis for the communications infrastructure. Tuples proved to be a convenient way of packaging RDF descriptions of collaboration events, and a pub/sub model was ideal for the framework. Furthermore, the persistent nature of tuples was useful for supporting late joining clients.

- The differing nature of the event types means that QoS requirements for the framework are non-trivial and would not be adequately handled by existing QoS frameworks. This is an area for further study.

8 Conclusions

This final chapter presents the overall conclusions from the thesis. It starts by giving a detailed breakdown of the contributions within this thesis and then discusses the potential for future work. Finally, open research issues that are relevant to this thesis are considered.

8.1 Contributions

The core contribution of this thesis has been the application of Semantic Web technologies to the domain of distributed real-time collaboration. This has been demonstrated by the development of a conceptual framework for automated live semantic annotation of distributed collaboration sessions, and a successful proof of concept implementation that was compliant to this framework. The remainder of this section gives a detailed chapter by chapter breakdown of the individual contributions that have gone to make up this overall outcome.

The literature review in chapter 2 identified a number of existing systems that supported some form of annotation of collaboration activities. It identified a number of desirable characteristics for systems of this type, which were support for machine processable semantics, live processing and automation. The review established that not one system fully provided all these features. A particular collective shortcoming exposed was a lack of machine processable semantics, limiting the scope for automated further processing. Additionally, the review of existing work in the domain of the Semantic Web exposed the fact that Semantic Web technologies have yet to be applied to the domain of the synchronous collaboration. The review also identified literature that described the concept of a mediated space and the potential for mediated interactions to be even more effective than face-to-face interactions.

Chapter 3 motivated the need for live semantic annotation of collaboration sessions. It did this by firstly identifying the benefits of providing additional information in the form of annotations and then discussing the advantage of a semantic approach, enabling significant potential for interoperability, reuse, extensibility and automation through inference.

A study of IRC use in telephone conferences was undertaken and it showed that live temporal annotation of collaboration sessions was useful to session participants, as was archiving the annotations for later use. The study was used along with experiences of using the Access Grid to establish a list of key event types that could make useful annotations and were common to many types of synchronous collaboration activities.

The problems of speaker identification and participant tracking in the Access Grid were identified and a scenario was described where these were overcome by semantic annotations to provide dynamically updated attendance lists and speaker highlighting. The scenario also described other annotation types and how they could be integrated other Semantic Web services. It was also established that these weaknesses of the Access Grid were common to other video and audio conferencing technologies and that they would also benefit from these forms of semantic annotations.

Chapter 4 developed the conceptual framework and the result was a general purpose architecture on which to base implementations of systems for performing real-time semantic annotation of live distributed collaboration sessions. It identified the role of producers for the generation of annotations and consumers as sinks for annotations, performing functions such as display. The need for an inference engine was established and it was argued that it would make sense for this to be a single component shared between sites in a given collaboration session. The need for a triplestore was also established to meet the dual requirements of providing additional knowledge for the inference process and providing storage for archiving annotations. The interaction mechanism between the producers and consumers was identified as needing to support communications that were pub/sub, real-time, reliable, multipoint and persistent and it was shown that these requirements map well to a tuple space.

It was established that live annotations needed to be represented as a pair of state change packets and it was shown how such packets could be packaged as tuples. The real-time requirements for the annotations were also discussed and it was argued that these were fairly flexible. It was also identified that due to the differing natures of the annotation and media streams, explicit synchronisation of these streams would be a very challenging task.

Chapter 5 described the implementation of a proof of concept system that was compliant to the conceptual architecture. The implementation was based on some of the functionality from the motivational scenario presented in chapter 3.

It identified the specific producers and consumers required and also developed a novel speaker identification technique for use in the Access Grid. The basic events that needed to be generated by the producers were determined and the inferences that could be made from them were identified. This was used to develop a detailed OWL ontology that formally specified the different annotations within the system and also enabled automatic entailments about location information. Furthermore, the effort in creating the system ontology was greatly reduced by identifying appropriate sections of existing ontologies and reusing them. Appropriate choices of existing software were also identified to provide the tuple space, inference engine and triplestore functions.

The implementation also demonstrated a display panel based consumer, which was capable of displaying the names of the connected sites and the names of the participants at those sites. This list updated in real-time to reflect changes to the makeup of the session and also to highlight the name of the current speaker.

Chapter 6 identified the logic required for the inferences, formally expressed as a set of inference rules, categorised by the type of operation the rules performed. The tests and actions that could not be performed by rules alone were established, and Jena builtins were created to implement these functions.

The categories of knowledge required to bootstrap the inference process for the proof of concept implementation were also determined and a small set of instances were created to enable the system to be tested. Reuse within the Semantic Web was demonstrated by obtaining some of this instance knowledge by querying the CS AKTiveSpace triplestore.

Finally, chapter 7 provided a discussion based evaluation of the conceptual framework and proof of concept implementation. A finding of particular interest was that the entailment features of Jena worked too slowly to allow real-time processing. This prompted the creation of a workaround for pre-computing certain entailments to enable real-time processing. Despite this limitation, the implementation demonstrated that real-time inferencing was practical.

The framework and implementation were found to have good support for inference, interoperability, reuse, automation and the chosen tools and technologies were found to be suitable. Missing or incorrect information in the triplestore was identified as a potential source for errors and techniques for minimising the impact of this were described.

8.2 Future Work

This section discusses a number of possible extensions to the work presented in this thesis.

8.2.1 Event Types and Inferences

One way of extending the work in this thesis would be to add further event types to the live collaboration ontology and create the appropriate producers, consumers and rules for handling them. Examples could be agenda items, slide transitions or hand authored text notes (e.g. taken on PDAs, see section 2.2.3). It would also be useful to try and harness any personal diary information (see section 2.4.4) to, for example, display a list of participants still expected to arrive.

Another useful inference would be to try and determine how formal or important a collaboration session is. This information could be used, for example, to display 'Do Not Disturb' on a screen outside the room of a formal meeting or could be used to automatically determine the level of intrusiveness [Ram04] that participants are willing to accept from sources such as mobile phones or Instant Messenger clients. Such a 'meeting importance' measure could possibly be derived from the job rank of the participants involved (which could be automatically obtained from the CS AKTive Space triplestore). For example in an academic setting, meetings primarily made up of postgraduate students and research assistants tend to be less formal (or more tolerant of interruptions) than those made up of higher ranking members of staff, like professors or heads of departments.

A further useful feature could be to extend the CS AKTiveSpace Communities of Practice work to flag to session participants which other people in the session are in their COP, and also show how they are linked to the individuals in their COP. Clearly this may often yield many uninteresting results, since people will usually already know who is working closely with them. However, the interesting results would be the ones where people share a COP but work in different fields or physical locations, as they may not be aware that they share a COP. It might be possible to automatically determine which results are interesting by looking for such indirect links between COP members.

One type of inferencing that has not been explored so far is offline inferencing after a meeting has ended to help improve the meeting archive. Offline inferencing could be used to perform inferences that are too complex to be performed in real-time or ones that simply need access to the entire meeting record. An example of offline inference would be to determine who the primary speaker in a meeting was, which could be used when searching meeting records. Furthermore, as the framework currently stands, there is no ability to exploit knowledge generated in previous meetings (for either real-time or offline inferencing). This might potentially be a valuable source of knowledge to feed into the inference process and is one that would be worthwhile exploring further.

8.2.2 Security

Another important infrastructure feature that so far has not been discussed is security. There may be times when a distributed meeting must be kept private so that unwanted 'snoopers' on the Internet cannot eavesdrop on the session content. Videoconferencing tools like Access Grid allow their media streams to be encrypted to prevent snooping, but clearly the addition of un-encrypted shared semantic annotations would be a weak point, giving the potential for snoopers to obtain some information about the session. Furthermore the pub/sub architecture means that any producer can join the tuple space and start publishing events. Clearly without any access controls an unauthorised producer could join the session and start publishing events which might either be unwanted or be deliberately incorrect to sabotage the session.

The most obvious solution to this would be to make the tuple space encrypted. This is a feature not currently supported by EQUIP. One way this could be achieved would be to extend EQUIP to support a symmetric encryption algorithm, meaning that all producers and consumers could only join a session's tuple space by using a pre-shared key. As it would be impractical for the operator at each site to configure each producer and consumer with the key, EQUIP's tuple space discovery mechanism could be extended to multicast the key (entered by the operator into the Session Information Producer) as well the dataspace URL. Although this multicasted information would only be local to each site, it could still be snooped, so this traffic could also be encrypted with a different pre-shared key, which each producer and consumer at a site could be pre-configured in a one off configuration step, in a similar way as they are already pre-configured to use a certain local multicast address for discovery.

8.2.3 Consumer Functionality

The proof of concept implementation only used a relatively simple consumer that passively displayed information to session participants. There is significant scope for extending consumer functionality to go beyond this. In particular, consumers could be created that ran on the personal laptops of session participants. This would be feasible as it is now commonplace for meeting rooms to be equipped with wireless networks and for participants to take laptops with them. Giving each participant a personal consumer would mean that the consumers could be interactive and could also personalise the information they displayed. An interactive consumer could, for example, provide a hypertext interface, allowing a participant to click on another participant's name and be presented with further information about that participant, such as contact details, job title or publications they have authored. All this information could be obtained by the consumer querying the CS AKTive Space triplestore. Personalisation could, for example, be used to highlight participants who are in the laptop owner's community of practice.

A drawback of potentially making consumers mobile is that the automated EQUIP dataspace discovery mechanism is not well suited to mobile devices, since the multicast address for discovery might be different in every meeting room. The most straightforward workaround for this would be to have the laptop user either manually enter the discovery address when joining the session (it could be displayed on the wall of the meeting room), or have them just enter the session dataspace URL directly.

The speaker identification functionality in the proof of concept application could also be further extended. As it currently stands, it only identifies the speaker by name and does not provide any additional cues to help identify which vic video window the speaker is located in. A useful extension would be to add window highlighting to vic, which could be used to easily draw attention to the window that contained the current speaker.

To highlight just the window that the speaker is depicted in is actually a very challenging task, since Access Grid sites typically transmit four simultaneous video feeds and determining which of those video window(s) the speaker is located is non trivial and would either require complex computer vision techniques or knowledge of where the camera is pointing combined with participant location information to determine who was in each shot.

A much simpler solution would be to highlight every video window from the site where the speaker is located, as this would still provide useful visual cues. This could be achieved by modifying vic to become a consumer. The vic feature for allowing individual video windows to be named could be used to specify the room URI for each video stream so that vic would know which windows to highlight. To ensure that the window names could still be read by humans, vic could also be modified to query the triplestore to resolve the room URIs into human readable names for display purposes.

8.3 Research Issues

This final section closes the thesis by discussing a number of open research issues relevant to this work.

8.3.1 Real Time Performance Issues

Testing of the proof of concept implementation revealed that Jena was not capable of computing its ontology based entailments in real-time. While not conclusive, this hints at a more fundamental potential problem for real-time Semantic Web applications. The complex interwoven nature of knowledge (at both instance and ontology level) within the Semantic Web could mean that real-time processing of this knowledge may at times be a challenging requirement. Ironically it is this potential for expressing complex interwoven relationships that is arguably one of the great strengths of the Semantic Web, enabling the Network Effect to answer new kinds of questions.

It could be argued that the inevitable increases in computing predicted by Moore's Law provide a straightforward solution to potential real-time performance problems. However, if the Semantic Web takes off, it is likely that the volume and complexity of knowledge it has to offer will continue to expand at an ever increasing rate, which could more than offset any increases in computing power.

The proof of concept implementation in this thesis overcame poor query performance by explicitly representing knowledge that otherwise would have only been implicit from the combination of the ontology and instance data. However, while this approach worked well for a relatively small scale application, such an approach would not scale well for the massive amounts of knowledge that the Semantic Web may one day contain. When dealing with knowledge on this scale, the total amount of implicit information becomes virtually limitless, making it totally infeasible to pre-compute and store. Therefore totally new techniques may have to be developed to enable the valuable implicit information in the Semantic Web to be harnessed in real-time.

8.3.2 Triplestore Architecture

The system architecture in this thesis was based around a single, centralised triplestore. This was a simple architecture to work with and is one that has been used in other applications such as CS AKTiveSpace. However this does not provide a massively scalable solution, and as De Roure and Hendler [DeR04] have stated, it is likely that the Semantic Web will evolve to use multiple triplestores and eventually many self-organising distributed RDF servers. Going beyond a single centrally managed triplestore is not only a challenge for architecture presented in this thesis, but also for the Semantic Web in general.

A particularly important issue is that of discovery; with a single triplestore, all components that need to use it can be simply pre-configured with its location. When the number of knowledge sources increases this becomes increasingly impractical, and when dealing with knowledge on the scale of the web, becomes virtually impossible.

A distributed Semantic Web introduces further problems of correctness and consistency. Without central management there is nothing to stop incorrect information being placed on the Semantic Web (either accidentally or deliberately), and with multiple knowledge sources it is also possible that contradicting knowledge may be published.

It was discussed in section 7.5.1 that the proof of concept implementation was not able to deal with incorrect or contradictory knowledge from the triplestore and at this stage it is not clear how it could be extended to robustly handle such cases. Resolving this issue may partly come down to trust. If knowledge is only taken from trusted sources, then it may potentially be more reliable than using un-trusted sources. This might be achieved through using digital signatures to sign statements, however as it would be difficult to explicitly specify all trusted sources, and it is likely that a 'web of trust' [Gol03] may emerge where, for example, if A trusts B and B trusts C, then A also can trust C, even though it is not explicitly stated.

Appendix A - Data From W3C Telephone Conferences

Title of WG	RDF Core WG					WebOnt WG					mean
date of meeting	30/08/02	23/08/02	16/08/02	09/08/02	26/07/02	26/09/02	19/09/02	12/09/02	05/09/02	29/08/02	
number of participants on phone	12	13	9	12	10	23	19	24	29	22	17.3
number of participatns in IRC	9	6	5	7	9	15	14	14	14	15	10.8
duration of teleconference	99	80	138	110	87	96	123	95	105	94	102.7
total number of IRC entries	278	365	255	573	376	288	172	431	158	447	334.3

Optional Zakim bot and RSSagent features used											total
list participants in telcon	x	x	x	x	x	x	x	x	x	x	10
manually identify telcon participants	x	x	x	x	x	x	x	x	x	x	10
muting telcon participant	x		x			x	x			x	5
identifying audio sources	x		x					x	x	x	5
disconnect telcon participant			x								1
dismiss Zakim bot											0
geographically locate dialing code											0
queuing to speak	x	x	x	x	x	x	x	x	x	x	10
speaker time limit											0
agenda tracking			x		x	x	x	x		x	6
future reminder ('ping')											0
scribe nomination			x		x	x	x	x		x	6
highlight action items			x		x	x					3
query for conference passcode		x					x				2
identify participants sharing phone	x			x							2

Misc data sent in IRC											total
comments	x	x	x	x	x			x	x	x	8
scribing	x	x	x	x	x			x	x	x	7
status (e.g.back in 5 mins)		x		x	x		x	x	x	x	7
URLs to agenda	x				x	x	x	x		x	6
URLs to email	x	x	x	x	x	x	x	x	x	x	10
URLs to misc. documents	x	x	x		x	x	x		x	x	8
agenda items	x	x	x	x	x	x	x	x		x	9
agenda item results	x	x		x	x	x		x		x	7
action items	x	x	x	x	x	x		x		x	8
discussing who will be scribe	x	x	x		x	x	x	x			7
indicating technical problems	x		x						x		3

Appendix B - IRC log of RDF Core Working Group Telcon

Log from telcon held on 16/08/2002

All names have been replaced with fictitious ones to ensure anonymity. The original log is publicly available on the web at http://www.w3.org/2002/08/16-rdfcore-irc

```
13:59:39 [RRSAgent]
        RRSAgent has joined #rdfcore
14:01:02 [jsmith]
        jsmith has joined #rdfcore
14:01:34 [MikeJones]
        MikeJones has joined #rdfcore
14:02:00 [MikeJones]
        Zakim, what's the passcode?
14:02:01 [Zakim]
        sorry, MikeJones, I don't know what conference this is
14:02:04 [MikeJones]
        Zakim, this is RDF
14:02:06 [Zakim]
        ok, MikeJones
14:02:12 [MikeJones]
        agenda?
14:02:34 [MikeJones]
        agenda + 16Aug http://lists.w3.org/Archives/Public/w3c-rdfcore-wg/2002Aug/0144.html
14:02:36 [ad]
        zakim, who is here?
14:02:37 [Zakim]
        On the phone I see ??P10, Davis, DaveW, GeorgeD, ??P14
14:02:38 [Zakim]
        On IRC I see MikeJones, jsmith, RRSAgent, Zakim, ad, rch, Adam, logger_1
14:03:07 [Zakim]
        +DaveP
14:03:15 [MikeJones]
        DaveP?
14:03:21 [Adam]
        zakim, DaveP is Adam
14:03:23 [Zakim]
        +Adam; got it
14:03:25 [MikeJones]
        ah
14:03:36 [Adam]
        zakim, who is muted
14:03:37 [Zakim]
        Adam, you need to end that query with '?'
14:03:38 [Zakim]
        +MikeJ
14:03:39 [Adam]
        zakim, who is muted?
14:03:40 [Zakim]
        I see no one muted
14:03:47 [ad]
        zakim, ??P10 is SimonR
14:03:48 [Zakim]
        +SimonR; got it
14:03:57 [ad]
        zakim, ??P14 is GavinK
```

```
14:03:58 [Zakim]
         +GavinK; got it
14:04:24 [Zakim]
         +John_Smith
14:04:37 [MikeJones]
         Zakim, pick a scribe
14:04:38 [Zakim]
         Not knowing who is chairing or who scribed recently, I propose MikeJ
14:04:40 [MikeJones]
         Zakim, pick a scribe
14:04:41 [Zakim]
         Not knowing who is chairing or who scribed recently, I propose DaveW
14:04:55 [jsmith]
         John Smith is scribing
14:05:01 [MikeJones]
         MikeJones has changed the topic to: RDFCore 16Aug. scribe: JohnS
14:05:14 [MikeJones]
         Zakim, who's on the phone?
14:05:15 [Zakim]
         On the phone I see SimonR, Davis, DaveW, GeorgeD, GavinK, Adam, MikeJ,
         John_Smith
14:05:16 [Adam]
         zakim, who is here?
14:05:17 [ad]
         zakim, who is here?
14:05:17 [Zakim]
         On the phone I see SimonR, Davis, DaveW, GeorgeD, GavinK, Adam, MikeJ,
         John_Smith
14:05:18 [Zakim]
         On the phone I see SimonR, Davis, DaveW, GeorgeD, GavinK, Adam, MikeJ,
         John_Smith
14:05:20 [Zakim]
         On IRC I see MikeJones, jsmith, RRSAgent, Zakim, ad, rch, Adam, logger_1
14:06:00 [jsmith]
         regrets: Pete, Mark, Anne Bolton, Nick, Alan Thompson, Dan Harris
14:06:29 [jsmith]
         Will Davis proposes WG sing happy birthday to Will
14:06:30 [Adam]
         zakim, mute Adam
14:06:31 [Zakim]
         Adam should now be muted
14:06:56 [jsmith]
         next telecon same time next week
14:07:08 [jsmith]
         no other agenda changes
14:07:21 [MikeJones]
         Zakim, pick a scribe
14:07:23 [ad]
         zakim, pick a victim.
14:07:23 [Zakim]
         Not knowing who is chairing or who scribed recently, I propose Davis
14:07:24 [Zakim]
         I don't understand 'pick a victim.', ad. Try /msg Zakim help
14:07:26 [MikeJones]
         Zakim, pick a scribe
14:07:28 [Zakim]
         Not knowing who is chairing or who scribed recently, I propose Davis
14:07:30 [MikeJones]
         Zakim, pick a scribe
14:07:32 [Zakim]
         Not knowing who is chairing or who scribed recently, I propose DaveW
14:07:34 [MikeJones]
```

```
          Zakim, pick a scribe
 14:07:35 [Zakim]
          Not knowing who is chairing or who scribed recently, I propose GavinK
 14:08:34 [MikeJones]
          Zakim, pick a scribe
 14:08:35 [Zakim]
          Not knowing who is chairing or who scribed recently, I propose DaveW
 14:08:37 [MikeJones]
          Zakim, pick a scribe
 14:08:38 [Zakim]
          Not knowing who is chairing or who scribed recently, I propose John_Smith
 14:08:40 [MikeJones]
          Zakim, pick a scribe
 14:08:42 [Zakim]
          Not knowing who is chairing or who scribed recently, I propose Adam (muted)
 14:08:44 [jsmith]
          Eric Miller will scribe next week if he's here (babysitting)
 14:09:05 [jsmith]
          reviewing action list
 14:09:29 [jsmith]
          minutes of July 19 missing?
 14:09:30 [Adam]
          zakim, unmute Adam
 14:09:31 [Zakim]
          Adam should no longer be muted
 14:10:05 [MikeJones]
          19 July IRC log: http://www.w3.org/2002/07/19-rdfcore-irc#T14-54-10
 14:10:35 [jsmith]
          Adam: split between 2 authors with IRC access vs. time
 14:10:55 [ad]
          http://www.w3.org/2002/07/19-rdfcore-irc.html
 14:10:57 [jsmith]
          IRC log was submitted and accepted as minutes
 14:11:40 [jsmith]
          approval of last week's minutes postponed due to late availability
 14:12:00 [jsmith]
          Brian will check IRC log against his action list
 14:12:27 [MikeJones]
          Zakim, who's talking?
 14:12:28 [Adam]
          zakim, drop Adam
 14:12:29 [Zakim]
          Adam is being disconnected
 14:12:29 [Zakim]
          -Adam
 14:12:31 [jsmith]
          reviewing long list state of completed actions
 14:12:37 [rch_]
          rch_ has joined #rdfcore
 14:12:38 [Zakim]
          MikeJones, listening for 10 seconds I heard sound from the following: MikeJ (29%)
 14:13:34 [jsmith]
          no objections to actions being closed off
 14:13:35 [MikeJones]
          === 8: daml:collection test case - volunteer to complete
 14:14:08 [jsmith]
          MikeJ has anyone implemented parseType="Literal"?
 14:14:11 [jsmith]
          rch: ARP has
 14:14:38 [jsmith]
          DaveW: validator produces ntriples
 14:14:59 [jsmith]
```

```
        ACTION: rch to complete test case
14:15:04 [Zakim]
        +Adam
14:15:38 [jsmith]
        item 9:
14:16:23 [jsmith]
        MikeJ: dark triple request from WebOnt may have gone away
14:16:45 [Adam]
        Should note that there is active discussion on www-rdf-logic about layering amongst
        Williams, Fuller and other WG members
14:16:51 [jsmith]
        ACTION: rch ask SWCG to check priority of dark triples requirement
14:16:55 [jsmith]
        item 10:
14:17:25 [jsmith]
        ad: wanted to confirm with rch as series editor
14:17:34 [jsmith]
        ad: additional pubrules cleanups, etc.
14:17:38 [MikeJones]
        validator looks buggy w.r.t. parseType="Literal"
14:17:46 [jsmith]
        rch: go ahead
14:18:34 [jsmith]
        ad: WG previously approved publication
14:18:45 [jsmith]
        ACTION: ad publish GUIDE
14:18:55 [jsmith]
        additional regrets: Stan
14:19:13 [jsmith]
        rch: on 19th, discussed detailed review of individual documents
14:19:26 [jsmith]
        ad: WG didn't get to this
14:20:17 [jsmith]
        ad: editors indicated that they would all need to include material on datatypes - more
        than an editorial change for last call - need to integrate and review schedule
14:20:40 [jsmith]
        ad: would require 2 reviews: current and with datatypes
14:22:01 [jsmith]
        Will: PRIMER doesn't yet address parseType="Collection" since it hasn't appeared in
        SYNTAX yet
14:22:42 [jsmith]
        Will: newer draft of July 25 on server now
14:23:14 [MikeJones]
        "on the server"... which server? where?
14:23:16 [jsmith]
        Adam: more work needed before review - would like another week
14:23:49 [MikeJones]
        #
14:23:50 [MikeJones]
        * new Primer version Will Davis (Thu, Jul 25 2002)
        http://lists.w3.org/Archives/Public/w3c-rdfcore-wg/2002Jul/0156.html
14:23:59 [jsmith]
        review SCHEMA week of August 30
14:24:08 [jsmith]
        PRIMER also August 30
14:25:01 [MikeJones]
        0156 -> http://www.w3.org/2001/09/rdfprimer/rdf-primer-20020725.html
14:25:49 [Zakim]
        +Chris_Moore
14:26:48 [jsmith]
        Chris: Model Theory waiting for data type resolution
```

Appendix C - The Live Collaboration Ontology

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE owl [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd "http://www.w3.org/2000/10/XMLSchema#">
  <!ENTITY dc "http://purl.org/dc/elements/1.1/">
  <!ENTITY dct "http://purl.org/dc/terms/">
  <!ENTITY support "http://www.aktors.org/ontology/support#">
  <!ENTITY portal "http://www.aktors.org/ontology/portal#">
  <!ENTITY meeting
"http://www.ecs.soton.ac.uk/~krp/coakting/rdf/meeting-20030606-2#">
  <!ENTITY location "http://signage.ecs.soton.ac.uk/location#">
  <!ENTITY live "http://www.ecs.soton.ac.uk/~bpj00r/ontologies/live-
meeting-20040319-1#">
  <!ENTITY base "http://www.ecs.soton.ac.uk/~bpj00r/ontologies/live-
meeting-20040319-1#">

]>

<rdf:RDF xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;"
       xmlns:dc="&dc;"
       xmlns:dct="&dct;"
         xmlns:xsd="&xsd;"
       xmlns:support="&support;"
       xmlns:portal="&portal;"
         xmlns:meeting="&meeting;"
         xmlns:live="&live;"
         xml:base="&base;">

  <owl:Ontology rdf:about="">
    <owl:versionInfo>1.2</owl:versionInfo>
    <rdfs:comment>The live collaboration ontology by Ben
Juby</rdfs:comment>
    <!-- import the CoAKTinG and Signage Location ontologies -->
    <!-- CoAKTinG ontology already imports the AKT ontologies, so no
need to explicitly import them -->
    <owl:imports
rdf:resource="http://www.ecs.soton.ac.uk/~krp/coakting/rdf/meeting-
20030606-2#"/>
    <owl:imports
rdf:resource="http://signage.ecs.soton.ac.uk/location#"/>
    <dc:creator>Ben Juby (bpj00r@ecs.soton.ac.uk)</dc:creator>
    <dct:created>2004-03-19</dct:created>
  </owl:Ontology>

  <!-- Personal Identifiers -->

  <owl:Class rdf:ID="Personal-Identifier">
    <rdfs:label>Personal Identifier</rdfs:label>
    <rdfs:comment>A generic class for tangible identifiers like
iButtons and RFID tags that uniquely identify a person</rdfs:comment>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:Class rdf:ID="iButton">
    <rdfs:label>iButton</rdfs:label>
    <rdfs:comment>A representation of an iButton</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Personal-Identifier"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="has-button-id">
    <rdfs:label>has button id</rdfs:label>
    <rdfs:comment>The 64 bit iButton ID represented as a string of
hexadecimal digits</rdfs:comment>
    <rdfs:domain rdf:resource="#iButton"/>
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:DatatypeProperty>

  <!-- Extend portal:Person to include a Personal-Identifier (eg an
iButton) -->

  <owl:ObjectProperty rdf:ID="has-personal-identifier">
    <rdfs:label>has personal identifier</rdfs:label>
    <rdfs:comment>a personal identifier (e.g. iButton) that uniquely
belongs to the person</rdfs:comment>
    <rdfs:domain rdf:resource="&portal;Person"/>
    <rdfs:range rdf:resource="#Personal-Identifier"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:ObjectProperty>

  <!-- new types of Event -->

  <owl:Class rdf:ID="Personal-Identifier-Event">
    <rdfs:label>Personal Identifier Event</rdfs:label>
    <rdfs:comment>Any event that involes a personal
identifier</rdfs:comment>
    <rdfs:subClassOf rdf:resource="&portal;Event"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="personal-identifier-used">
    <rdfs:label>personal identifier used</rdfs:label>
    <rdfs:comment>Records the instance of personal identifier used in
the event, if known</rdfs:comment>
    <rdfs:domain rdf:resource="#Personal-Identifier-Event"/>
    <rdfs:range rdf:resource="#Personal-Identifier"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:ObjectProperty>

  <owl:Class rdf:ID="iButton-Signed-In">
    <rdfs:label>iButton Signed In</rdfs:label>
    <rdfs:comment>Represents an iButton being 'signed
in'</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Personal-Identifier-Event"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&portal;has-location" />
        <owl:allValuesFrom rdf:resource="#iButton-Reader-Position" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="id-of-ibutton-used">
    <rdfs:label>id of ibutton used</rdfs:label>
    <rdfs:comment>Records the 64 bit hex ID of the iButton used in the
event</rdfs:comment>
    <rdfs:domain rdf:resource="#iButton-Signed-In"/>
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:DatatypeProperty>

  <owl:Class rdf:ID="Generic-Agent-Present">
    <rdfs:label>Generic Agent Present</rdfs:label>
    <rdfs:comment>Represents a Generic Agent being present in a
collaboration session</rdfs:comment>
    <rdfs:subClassOf rdf:resource="&portal;Event"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:Class rdf:ID="Legal-Agent-Present">
    <rdfs:label>Legal Agent Present</rdfs:label>
    <rdfs:comment>Represents a Legal Agent being present in a
collaboration session</rdfs:comment>
    <rdfs:subClassOf rdf:resource="Generic-Agent-Present"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&portal;has-main-agent" />
        <owl:allValuesFrom rdf:resource="&portal;Legal-Agent" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:Class rdf:ID="Person-Present">
    <rdfs:label>Person Present</rdfs:label>
    <rdfs:comment>Represents a Person being present in a collaboration
session</rdfs:comment>
    <rdfs:subClassOf rdf:resource="Legal-Agent-Present"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&portal;has-main-agent" />
        <owl:allValuesFrom rdf:resource="&portal;Person" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>
  <owl:Class rdf:ID="Microphone-Active">
    <rdfs:label>Microphone Active</rdfs:label>
    <rdfs:comment>An event to indicate that a microphone is gated
on</rdfs:comment>
    <rdfs:subClassOf rdf:resource="&portal;Event"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&portal;has-location" />
        <owl:allValuesFrom rdf:resource="#Microphone-Position" />
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:Class rdf:ID="Joined-To-Session">
    <rdfs:label>Joined To Session</rdfs:label>
    <rdfs:comment>An event that represents when a physical location is
currently joined to a distributed collaboration session</rdfs:comment>
    <rdfs:subClassOf rdf:resource="&portal;Event"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <!-- Extensions to the Signage Location ontology -->

  <!-- Extend location:Abstract-Space to say that it is equivalent to
portal:Location -->

  <rdf:Description rdf:about="&location;Abstract-Space">
    <owl:equivalentClass rdf:resource="&portal;Location"/>
  </rdf:Description>

  <!-- Declare location:is-located-in property as transitive -->

  <rdf:Description rdf:about="&location;is-located-in">
    <rdf:type rdf:resource="&owl;TransitiveProperty" />
  </rdf:Description>

  <!-- Declare location:adjacent-to as owl:SymmetricProperty -->

  <rdf:Description rdf:about="&location;adjacent-to">
    <rdf:type rdf:resource="&owl;SymmetricProperty" />
```

```
   </rdf:Description>

   <!-- Extend location:Abstract-Space to have a property that records
 a collaboration site name -->

   <owl:DatatypeProperty rdf:ID="has-collaboration-site-name">
     <rdfs:label>has collaboration site name</rdfs:label>
     <rdfs:comment>The name of the site that is shown to people at
 others sites - e.g. "Southampton ECS"</rdfs:comment>
     <rdfs:domain rdf:resource="&location;Abstract-Space"/>
     <rdfs:range rdf:resource="&xsd;string"/>
     <rdfs:isDefinedBy rdf:resource="&base;"/>
   </owl:DatatypeProperty>

   <!-- new types of location:Abstract-Space -->

   <owl:Class rdf:ID="Device-Position">
     <rdfs:label>Device-Position</rdfs:label>
     <rdfs:comment>the location of any device</rdfs:comment>
     <rdfs:subClassOf rdf:resource="&location;Abstract-Space" />
   </owl:Class>

   <owl:Class rdf:ID="iButton-Reader-Position">
     <rdfs:label>iButton-Reader-Position</rdfs:label>
     <rdfs:comment>the location of an iButton reader</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Device-Position" />
   </owl:Class>

   <owl:Class rdf:ID="Microphone-Position">
     <rdfs:label>Microphone-Position</rdfs:label>
     <rdfs:comment>the location of a microphone</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Device-Position" />
   </owl:Class>

   <owl:Class rdf:ID="Seating-Position">
     <rdfs:label>Seating-Position</rdfs:label>
     <rdfs:comment>a specific seating position in a meeting
room</rdfs:comment>
     <rdfs:subClassOf rdf:resource="&location;Abstract-Space" />
   </owl:Class>

</rdf:RDF>
```

Appendix D - The CoAKTinG Ontology

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE owl [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd "http://www.w3.org/2000/10/XMLSchema#">
  <!ENTITY dc "http://purl.org/dc/elements/1.1/">
  <!ENTITY dct "http://purl.org/dc/terms/">
  <!ENTITY support "http://www.aktors.org/ontology/support#">
  <!ENTITY portal "http://www.aktors.org/ontology/portal#">
  <!-- ENTITY meeting "http://www.aktors.org/ontology/meeting#" -->
  <!ENTITY meeting
"http://www.ecs.soton.ac.uk/~krp/coakting/rdf/meeting-20030606-2#">
  <!-- ENTITY base "http://www.aktors.org/ontology/meeting" -->
  <!ENTITY base "http://www.ecs.soton.ac.uk/~krp/coakting/rdf/meeting-
20030606-2">
]>

<!-- CoAKTinG meeting ontology added above, and as namespace below -->

<rdf:RDF xmlns:owl="&owl;"
         xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;"
       xmlns:dc="&dc;"
       xmlns:dct="&dct;"
         xmlns:xsd="&xsd;"
       xmlns:support="&support;"
       xmlns:portal="&portal;"
         xmlns:meeting="&meeting;"
         xml:base="&base;">

  <owl:Ontology rdf:about="&base;">
    <rdfs:label xml:lang="en">CoAKTinG Meeting Ontology.</rdfs:label>
    <dc:title xml:lang="en">CoAKTinG Meeting Ontology.</dc:title>
    <dc:description xml:lang="en">The CoAKTinG Meeting Ontology has
been designed to support the CoAKTinG project and tools, extending the
AKT Reference Ontology.</dc:description>
    <dc:creator>CoAKTinG Project</dc:creator>
    <dc:creator>Kevin R. Page</dc:creator>
```

```
      <dct:created>2003-06-02</dct:created>
      <owl:versionInfo>0.2</owl:versionInfo>
      <owl:imports
 rdf:resource="http://www.aktors.org/ontology/portal"/>
    </owl:Ontology>

    <!-- add milliseconds to TimePoints -->

    <owl:DatatypeProperty rdf:ID="millisecond-of">
      <rdfs:label xml:lang="en">millisecond of</rdfs:label>
      <rdfs:domain rdf:resource="&support;Time-Point"/>
      <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
      <rdfs:isDefinedBy rdf:resource="&base;"/>
    </owl:DatatypeProperty>

    <owl:Class rdf:about="&support;Time-Point">
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#millisecond-of"/>
          <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>

    <!-- a new subsclass of Event to represent meetings which
         concurrently take place in several locations -->

    <owl:Class rdf:ID="Distributed-Gathering">
      <rdfs:label>Distributed Gathering</rdfs:label>
      <rdfs:comment>Gatherings that take place in more than one physical
location.</rdfs:comment>
      <rdfs:subClassOf rdf:resource="&portal;Social-Gathering"/>
      <rdfs:isDefinedBy rdf:resource="&base;"/>

      <!-- a Distributed Gathering must have one or more constituent
           Events -->
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#has-local-event"/>
          <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
```

```
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="has-local-event">
    <rdfs:subPropertyOf rdf:resource="&portal;has-sub-event"/>
    <rdfs:domain rdf:resource="#Distributed-Gathering"/>
    <rdfs:range rdf:resource="&portal;Social-Gathering"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:ObjectProperty>

  <!-- Information-Exhibition is a subclass of Information-Transfer-
Event, which
       is used to express the exhibition / display of an Information-
Bearing-Object,
       e.g. the presentation of slides or documents in a meeting -->

  <owl:Class rdf:ID="Information-Exhibition">
    <rdfs:label>Information Exhibition</rdfs:label>
    <rdfs:comment>Information Exhibition expresses the display of an
Information-Bearing-Object, e.g. the presentation of slides or
documents in a meeting.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="&portal;Information-Transfer-
Event"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="has-information-object">
    <rdfs:domain rdf:resource="#Information-Exhibition"/>
    <rdfs:range rdf:resource="&portal;Information-Bearing-Object"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:ObjectProperty>

  <!-- Define a class to describe Compound Information Objects, e.g. a
       presentation that includes multiple slides, video etc. -->

  <owl:Class rdf:ID="Compound-Information-Object">
    <rdfs:label>Compound Information Object</rdfs:label>
    <rdfs:comment>Compound Information Objects describe Information
Bearing Objects that are constructed from a collection of further
Information Bearing Objects. e.g. a presentation containing several
slides and a video.</rdfs:comment>
```

```
    <rdfs:subClassOf rdf:resource="&portal;Information-Bearing-
Object"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="has-component">
    <rdfs:domain rdf:resource="#Compound-Information-Object"/>
    <rdfs:range rdf:resource="&portal;Information-Bearing-Object"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:ObjectProperty>

  <owl:Class rdf:ID="Presentation">
    <rdfs:label>Presentation</rdfs:label>
    <rdfs:comment>e.g. a PowerPoint presentation</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Compound-Information-Object"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:Class rdf:ID="Slide">
    <rdfs:label>Slide</rdfs:label>
    <rdfs:comment>A slide within a presentation</rdfs:comment>
    <rdfs:subClassOf rdf:resource="&portal;Information-Bearing-
Object"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="has-rendered-uri">
    <rdfs:label xml:lang="en">has rendering</rdfs:label>
    <rdfs:comment xml:lang="en">The location of a rendering of an
Information Bearing Object. e.g. a JPEG rendering of a
Slide.</rdfs:comment>
    <rdfs:domain rdf:resource="&portal;Information-Bearing-Object"/>
    <!-- rdfs:range rdf:resource="&portal;Information-Bearing-Object"/
-->
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:ObjectProperty>

  <!-- Verbal comment Event -->

  <owl:Class rdf:ID="Making-a-Verbal-Comment">
    <rdfs:label>Verbal Comment</rdfs:label>
    <rdfs:comment>An Event to bind when a Person makes a comment (e.g.
 in a meeting).</rdfs:comment>
    <rdfs:subClassOf rdf:resource="&portal;Information-Transfer-
 Event"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&portal;sender-of-information"
/>
        <owl:allValuesFrom rdf:resource="&portal;Person" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="has-transcription">
    <rdfs:label xml:lang="en">has transcription</rdfs:label>
    <rdfs:comment xml:lang="en">The transcription of an event, e.g.
the minutes of a meeting, or the video recording of a
presentation.</rdfs:comment>
    <rdfs:domain rdf:resource="&portal;Event"/>
    <rdfs:range rdf:resource="&portal;Information-Bearing-Object"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:ObjectProperty>


  <!-- The has-start-time and has-end-time are expected to map to the
creation and last
       modified times of Compendium nodes -->

  <owl:Class rdf:ID="Creating-a-Compendium-Node">
    <rdfs:label xml:lang="en">creating a compendium node</rdfs:label>
    <rdfs:comment xml:lang="en">This event marks when a Compendium
node was created e.g. when compendium is used to minute a
meeting.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="&portal;Information-Transfer-
Event"/>
    <rdfs:isDefinedBy rdf:resource="&base;"/>

    <rdfs:subClassOf>
      <owl:Restriction>
```

```
        <owl:onProperty rdf:resource="&portal;sender-of-information"
/>
        <owl:allValuesFrom rdf:resource="&portal;Person" />
      </owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&meeting;has-compendium-node" />
        <owl:allValuesFrom rdf:resource="&portal;Person" />
        <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>

  </owl:Class>

  <owl:ObjectProperty rdf:ID="has-compendium-node">
    <rdfs:label xml:lang="en">has Compendium node</rdfs:label>
    <rdfs:comment xml:lang="en">A Compendium node being created.
Currently the resource is expected to be within the XML output from
Compendium, rather than a class/instance in the knowledge
base.</rdfs:comment>
    <rdfs:domain rdf:resource="&meeting;Creating-a-Compendium-Node"/>
    <!-- rdfs:range rdf:resource="&meeting;Agumentation"/ -->
    <rdfs:isDefinedBy rdf:resource="&base;"/>
  </owl:ObjectProperty>

</rdf:RDF>
```

Appendix E - The Signage Location Ontology

```
<?xml version="1.0"?>

<!DOCTYPE owl [
  <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY owl  "http://www.w3.org/2002/07/owl#">
  <!ENTITY dc   "http://purl.org/dc/elements/1.1/">
  <!ENTITY dct  "http://purl.org/dc/terms/">
  <!ENTITY akt  "http://www.aktors.org/ontology/portal#">
  <!ENTITY akts "http://www.aktors.org/ontology/support#">
  <!ENTITY base "http://signage.ecs.soton.ac.uk/location#">
]>

<rdf:RDF xmlns:rdf="&rdf;"
         xmlns:rdfs="&rdfs;"
         xmlns:xsd="&xsd;"
         xmlns:owl="&owl;"
         xmlns:dc="&dc;"
         xmlns:dct="&dct;"
         xmlns:akt="&akt;"
         xmlns:akts="&akts;"
         xml:base="&base;">

  <owl:Ontology rdf:about="">
    <rdfs:label>Building Ontology</rdfs:label>
    <dc:title xml:lang="en">Building Ontology</dc:title>
    <dc:description xml:lang="en">The Building Ontology has been
designed to describe the structure, contents and occupants of a
building, in order support a number of pervasive computing
applications.</dc:description>
    <dc:creator>Signage Project
(http://signage.ecs.soton.ac.uk/)</dc:creator>
    <dc:creator>Ian Millard (icm02r@ecs.soton.ac.uk)</dc:creator>
    <dct:created>2004-01-19</dct:created>
    <owl:versionInfo>0.1</owl:versionInfo>
  </owl:Ontology>
```

```
<!-- Abstract space, and associated properties -->

<owl:Class rdf:ID="Abstract-Space">
  <rdfs:label>Abstract-Space</rdfs:label>
  <rdfs:comment>This is a high-level abstraction of any abstract
space</rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:ID="is-located-in">
  <rdfs:label>is-located-in</rdfs:label>
  <rdfs:comment>This property is to be used to describe an Abstract-
Space which is located within another,</rdfs:comment>
  <rdfs:domain rdf:resource="#Abstract-Space" />
  <rdfs:range rdf:resource="#Abstract-Space" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="is-part-of">
  <rdfs:label>is-part-of</rdfs:label>
  <rdfs:comment>This property is to be used to describe an Abstract-
Space which forms part of another</rdfs:comment>
  <rdfs:domain rdf:resource="#Abstract-Space" />
  <rdfs:range rdf:resource="#Abstract-Space" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="is-owned-by">
  <rdfs:label>is-owned-by</rdfs:label>
  <rdfs:comment>This property is to be used to describe the owning
Organization of the Abstract-Space</rdfs:comment>
  <rdfs:domain rdf:resource="#Abstract-Space" />
  <rdfs:range rdf:resource="&akt;Organization" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has-usual-occupant">
  <rdfs:label>has-usual-occupant</rdfs:label>
  <rdfs:comment>This property is to be used to describe the usual
occupant of an Abstract-Space</rdfs:comment>
  <rdfs:domain rdf:resource="#Abstract-Space" />
  <rdfs:range rdf:resource="&akt;Person" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="adjacent-to">
```

```
      <rdfs:label>adjacent-to</rdfs:label>
      <rdfs:comment>This property indicates that one Abstract Space is
 adjacent to another</rdfs:comment>
      <rdfs:domain rdf:resource="#Abstract-Space" />
      <rdfs:range  rdf:resource="#Abstract-Space" />
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:ID="adjacent-on-north-side">
      <rdfs:label>adjacent-on-north-side</rdfs:label>
      <rdfs:comment>This property is to be used to describe another
 adjacent Abstract Space which is to the North</rdfs:comment>
      <rdfs:subPropertyOf rdf:resource="#adjacent-to" />
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:ID="adjacent-on-south-side">
      <rdfs:label>adjacent-on-south-side</rdfs:label>
      <rdfs:comment>This property is to be used to describe another
 adjacent Abstract Space which is to the South</rdfs:comment>
      <rdfs:subPropertyOf rdf:resource="#adjacent-to" />
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:ID="adjacent-on-east-side">
      <rdfs:label>adjacent-on-east-side</rdfs:label>
      <rdfs:comment>This property is to be used to describe another
 adjacent Abstract Space which is to the East</rdfs:comment>
      <rdfs:subPropertyOf rdf:resource="#adjacent-to" />
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:ID="adjacent-on-west-side">
      <rdfs:label>adjacent-on-west-side</rdfs:label>
      <rdfs:comment>This property is to be used to describe another
 adjacent Abstract Space which is to the West</rdfs:comment>
      <rdfs:subPropertyOf rdf:resource="#adjacent-to" />
    </owl:ObjectProperty>

    <!-- Enclosed space, and associated properties -->

    <owl:Class rdf:ID="Enclosed-Space">
      <rdfs:label>Enclosed-Space</rdfs:label>
      <rdfs:comment>This is a high-level abstraction of any enclosed or
 bounded space (such as a building or room) which constrians movement
 from one space to another</rdfs:comment>
```

```
    <rdfs:subClassOf rdf:resource="#Abstract-Space" />
  </owl:Class>

  <owl:ObjectProperty rdf:ID="permits-access-to">
    <rdfs:label>permits-access-to</rdfs:label>
    <rdfs:comment>This property is to be used to describe a connection
(such as a door) which permits access between two Enclosed-
Spaces</rdfs:comment>
    <rdfs:domain rdf:resource="#Enclosed-Space" />
    <rdfs:range rdf:resource="#Abstract-Space" />
  </owl:ObjectProperty>

  <!-- A building -->

  <owl:Class rdf:ID="Building">
    <rdfs:label>Building</rdfs:label>
    <rdfs:comment>This class is used to represent a
Building</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Enclosed-Space" />
  </owl:Class>

  <owl:ObjectProperty rdf:ID="has-postal-address">
    <rdfs:label>has-postal-address</rdfs:label>
    <rdfs:comment>This property denotes that a Building is located at
a particular Postal-Address</rdfs:comment>
    <rdfs:domain rdf:resource="#Building" />
    <rdfs:range  rdf:resource="&akt;Postal-Address" />
  </owl:ObjectProperty>

  <!-- A floor in a building -->

  <owl:Class rdf:ID="Floor">
    <rdfs:label>Floor</rdfs:label>
    <rdfs:comment>This class is used to represent a Floor in a
Building</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Enclosed-Space" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#is-part-of" />
        <owl:allValuesFrom rdf:resource="#Building" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <!-- A Room on a Floor of a Building -->

  <owl:Class rdf:ID="Room">
    <rdfs:label>Room</rdfs:label>
    <rdfs:comment>This class is used to represent a Room on a Floor in
a Building</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Enclosed-Space" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#is-part-of" />
        <owl:allValuesFrom rdf:resource="#Floor" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <!-- Types of room -->

  <owl:Class rdf:ID="Office">
    <rdfs:label>Office</rdfs:label>
    <rdfs:comment>This class is used to represent an Office, usually
inhabited by a small number of people</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Room" />
  </owl:Class>

  <owl:Class rdf:ID="Laboratory">
    <rdfs:label>Office</rdfs:label>
    <rdfs:comment>This class is used to represent a Laboratory,
usually inhabited by a large number of people</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Room" />
  </owl:Class>

  <owl:Class rdf:ID="Meeting-Room">
    <rdfs:label>Meeting-Room</rdfs:label>
    <rdfs:comment>This class is used to represent a room used for
meetings</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Room" />
  </owl:Class>

  <!-- A  Corridoor -->
```

```
   <owl:Class rdf:ID="Corridoor">
     <rdfs:label>Corridoor</rdfs:label>
     <rdfs:comment>This class is used to represent a corridoor, on a
 Floor of a Building</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Enclosed-Space" />
     <rdfs:subClassOf>
       <owl:Restriction>
         <owl:onProperty rdf:resource="#is-part-of" />
         <owl:allValuesFrom rdf:resource="#Floor" />
       </owl:Restriction>
     </rdfs:subClassOf>
   </owl:Class>

   <!-- Floor-Traversing-Spaces -->

   <owl:Class rdf:ID="Floor-Traversing-Space">
     <rdfs:label>Floor-Traversing-Space</rdfs:label>
     <rdfs:comment>This class is used to represent a space which
 traverses Floors</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Enclosed-Space" />
     <rdfs:subClassOf>
       <owl:Restriction>
         <owl:onProperty rdf:resource="#is-part-of" />
         <owl:allValuesFrom rdf:resource="#Building" />
       </owl:Restriction>
     </rdfs:subClassOf>
   </owl:Class>

   <owl:Class rdf:ID="Stairs">
     <rdfs:label>Stairs</rdfs:label>
     <rdfs:comment>This class is used to represent Stairs, which
 traverse between Floors</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Floor-Traversing-Space" />
   </owl:Class>

   <owl:Class rdf:ID="Lift">
     <rdfs:label>Lift</rdfs:label>
     <rdfs:comment>This class is used to represent a lift, which
 traverses between Floors</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Floor-Traversing-Space" />
   </owl:Class>
```

```
  <!-- A Partitioned-Space -->

  <owl:Class rdf:ID="Partitioned-Space">
    <rdfs:label>Partitioned-Space</rdfs:label>
    <rdfs:comment>This class is used to represent a partitioned space
in a Building. This is an Enclosed-Space, like a room, but which may
permit communication between Partitioned-Spaces</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Enclosed-Space" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#is-part-of" />
        <owl:allValuesFrom rdf:resource="#Enclosed-Space" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <!-- Work area -->

  <owl:Class rdf:ID="Work-Area">
    <rdfs:label>Work-Area</rdfs:label>
    <rdfs:comment>This class is used to represent a localised area in
which someone works, such as a desk, laboratory bench, machine in a
workshop etc</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Abstract-Space" />
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#is-located-in" />
        <owl:allValuesFrom rdf:resource="#Abstract-Space" />
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

</rdf:RDF>
```

Appendix F - The Inference Rules

1 - Get Locations On Sign In

```
[getLocationsOnSignIn:

//query for the things that an iButton reader is-located-in
(?sign_in rdf:type live:iButton-Signed-In),
(?sign_in portal:has-location ?reader_loc),
noValue(?reader_loc, location:is-located-in)

->

print("getLocationsOnSignIn has fired"),
queryTriplestore(?reader_loc, location:is-located-in, ?loc)
]
```

2 - Get Locations On Microphone Active

```
[getLocationsOnMicrophoneActive:

//query for the things that a microphone position is-located-in
(?mic_active rdf:type live:Microphone-Active),
(?mic_active portal:has-location ?mic_pos),
noValue(?mic_pos, location:is-located-in)

->

print("getLocationsOnMicrophoneActive has fired" ),
queryTriplestore(?mic_pos, location:is-located-in, ?loc)
]
```

3 - iButton ID To URI

```
[iButtonIDToURI:

//resolves an iButton ID to its URI
(?a live:id-of-ibutton-used ?id)

->

print("iButtonIDToURI has fired"),
```

3 continued:

```
queryTriplestore(?ibutton, live:has-ibutton-id, ?id),
]
```

4 - iButton To Person

```
[iButtonToPerson:

//resolves the URI of an iButton to a person
(?ibutton rdf:type live:iButton)

->

print("iButtonToPerson has fired "),
queryTriplestore (?person, live:has-personal-identifier, ?ibutton)
]
```

5 - Create Person Present

```
[createPersonPresent:

(?a rdf:type live:iButton-Signed-In),
(?a portal:has-location ?location),
(?a live:id-of-ibutton-used ?id),

(?a support:has-time-interval ?time_int),
(?time_int support:begins-at-time-point ?begin_time),

(?person live:has-personal-identifier ?ibutton),
(?ibutton live:has-ibutton-id ?id),

makeResource(?pp_event),
makeResource(?time)

->

print("createPersonPresent has fired"),
(?pp_event, rdf:type live:Person-Present),
(?pp_event, portal:has-location ?location),

(?pp_event support:has-time-interval ?time),
(?time rdf:type support:Time-Interval),
(?time support:begins-at-time-point ?begin_time),
```

5 continued:

```
(?pp_event, portal:has-main-agent ?person),

publishToDataspace("TUPLE_TYPE", "TUPLE"),
publishToDataspace("EVENT_TYPE", live:Person-Present),

publishToDataspace("ADD_TRIPLE", ?pp_event, rdf:type, live:Person-
Present),
publishToDataspace("ADD_TRIPLE", ?pp_event, portal:has-location,
?location),

publishToDataspace("ADD_TRIPLE", ?pp_event support:has-time-interval
?time),
publishToDataspace("ADD_TRIPLE", ?time rdf:type support:Time-
Interval),
publishToDataspace("ADD_TRIPLE", ?time support:begins-at-time-point
?begin_time),
publishToDataspace("ADD_TRIPLE", ?pp_event, portal:has-main-agent
?person),

publishToDataspace("PUBLISH")
]
```

6 - Create Single Meeting In One Room

```
[createSingleMeetingInOneRoom:

(?a rdf:type live:Person-Present),
(?b rdf:type live:Person-Present),
notEqual(?a,?b),

//check that both events have the same room location
//this location should be an ibutton reader
(?a portal:has-location ?loc_a),
(?b portal:has-location ?loc_b),

//3store will have already been queried when getLocations fired
(?loc_a location:is-located-in ?room),
(?loc_b location:is-located-in ?room),
(?room rdf:type location:Meeting-Room),

noMeetingAtPhysLoc(?room),
```

6 continued:

```
//check that both person present events are not in a meeting
eventNotInMeeting(?a),
eventNotInMeeting(?b),

//get time intervals and start times
(?a support:has-time-interval ?time_a),
(?b support:has-time-interval ?time_b),
(?time_a support:begins-at-time-point ?begin_a),
(?time_b support:begins-at-time-point ?begin_b),

//check that the PP events are still active
noValue(?time_a support:ends-at-time-point),
noValue(?time_b support:ends-at-time-point),

getMostRecentTimePoint(?begin_a, ?begin_b, ?most_recent),

//get 'main agents'
(?a portal:has-main-agent ?person_a),
(?b portal:has-main-agent ?person_b),

makeResource(?meeting),
makeResource(?time),

//check that this rule hasn't fired before
//but with data the other way round
noValue (?b live:csmior-has-fired),

->

print("createSingleMeetingInOneRoom has fired"),

//create a Meeting-Taking-Place

//add person to meeting rule will
//then fire to add the Person-Present events

//create a new meeting taking place
(?meeting rdf:type portal:Meeting-Taking-Place),

//create appropriate start time
(?meeting support:has-time-interval ?time),
```

6 continued:

```
(?time rdf:type support:Time-Interval),

//I want the most recent sign in - need to test
(?time support:begins-at-time-point ?most_recent),

//also need to add room location
(?meeting portal:has-location ?room),

//assert some a unique triple to say rule has fired
(?a live:csmior-has-fired "fired")

]
```

7 - Create Meetings In Two Rooms

```
[createMeetingsInTwoRooms:

(?a rdf:type live:Person-Present),
(?b rdf:type live:Person-Present),
notEqual(?a,?b),

//check that the room locations are different

//this location should be an ibutton reader
(?a portal:has-location ?loc_a),
(?b portal:has-location ?loc_b),

//3store will have already been queried when getLocations fired
(?loc_a location:is-located-in ?room_a),
(?loc_b location:is-located-in ?room_b),
(?room_a rdf:type location:Meeting-Room),
(?room_b rdf:type location:Meeting-Room),
notEqual(?room_a,?room_b),

noMeetingAtPhysLoc(?room_a),
noMeetingAtPhysLoc(?room_b),

//check that both person present events are not in a meeting
//this test is needed because if Person-Present event a or b
//were already part of meetings that were finished then this
//rule would still fire
eventNotInMeeting(?a),
```

7 continued:

```
eventNotInMeeting(?b),

//get time intervals and start times
(?a support:has-time-interval ?timeint_a),
(?b support:has-time-interval ?timeint_b),
(?timeint_a support:begins-at-time-point ?begin_a),
(?timeint_b support:begins-at-time-point ?begin_b),

//check that the PP events are still active
noValue(?timeint_a support:ends-at-time-point)
noValue(?timeint_b support:ends-at-time-point)

getMostRecentTimePoint(?begin_a, ?begin_b, ?most_recent),

//get 'main agents'
(?a portal:has-main-agent ?person_a),
(?b portal:has-main-agent ?person_b),



//create new resouces to become meetings and time intervals
makeResource(?meeting_a),
makeResource(?meeting_b),

makeResource(?time_a),
makeResource(?time_b),

//check that this rule hasn't fired before
//but with data the other way round
noValue (?b live:cmitr-has-fired)

->

print("createMeetingsInTwoRooms has fired"),

//create a 2 instances of Meeting-Taking-Place

//add person to meeting rule will
//then fire to add the Person-Present events

//create 2 new meetings taking place
(?meeting_a rdf:type portal:Meeting-Taking-Place),
```

7 continued:

```
(?meeting_b rdf:type portal:Meeting-Taking-Place),

//create appropriate start times
(?meeting_a support:has-time-interval ?time_a),
(?meeting_b support:has-time-interval ?time_b),

(?time_a rdf:type support:Time-Interval),
(?time_b rdf:type support:Time-Interval),

//I want the most recent sign in - need to test
(?time_a support:begins-at-time-point ?most_recent),
(?time_b support:begins-at-time-point ?most_recent),

//add the agent present as sub events of the meetings
//(?meeting_a portal:has-sub-event ?a),
//(?meeting_b portal:has-sub-event ?b),

//add participants to the social gathering
//(?meeting_a portal:meeting-attendee person_a),
//(?meeting_b portal:meeting-attendee person_b),

//also need to add room location
(?meeting_a portal:has-location ?room_a),
(?meeting_b portal:has-location ?room_b),

//assert some a unique triple to say rule has fired
(?a live:cmitr-has-fired "fired")

]
```

8 - Add Person To Meeting

```
[addPersonToMeeting:

(?a rdf:type live:Person-Present),

//this location should be an ibutton reader
(?a portal:has-location ?loc),

//3store will have already been queried when getLocations fired
(?loc location:is-located-in ?room),
(?room rdf:type location:Meeting-Room),
```

8 continued:

```
//match on a meeting currently in session in the meeting room
(?meeting rdf:type portal:Meeting-Taking-Place),
(?meeting portal:has-location ?room),
(?meeting support:has-time-interval ?time),
noValue(?time support:ends-at-time-point),

//check that the person present event is not already in a meeting
eventNotInMeeting(?a),

//check that pp event is still active
(?a support:has-time-interval ?pptime),
noValue(?pptime support:ends-at-time-point),

//get 'main agent'
(?a portal:has-main-agent ?person),

->

print("addPersonToMeeting has fired"),

//add the person present to the meeting
(?meeting portal:has-sub-event ?a),

//add participant to the meeting
(?meeting portal:meeting-attendee person)

]
```

9 - Create Distributed Gathering

```
[createDistributedGathering:

//match on two different meetings taking place
(?meeting_a rdf:type portal:Meeting-Taking-Place),
(?meeting_b rdf:type portal:Meeting-Taking-Place),
notEqual(?meeting_a,?meeting_b),

//check there is not already an instance of Distributed Gathering
noValue3(?existing_dist_gath, rdf:type, meeting:Distributed-
Gathering),
```

9 continued:

```
//get time intervals and start times
(?meeting_a support:has-time-interval ?time_a),
(?meeting_b support:has-time-interval ?time_b),
(?time_a support:begins-at-time-point ?begin_a),
(?time_b support:begins-at-time-point ?begin_b),

//check that the meetings are still in session
noValue(?time_a support:ends-at-time-point),
noValue(?time_b support:ends-at-time-point),

getMostRecentTimePoint(?begin_a, ?begin_b, ?most_recent),

makeResource(?dist_gath),

makeResource(?time),

//check that this rule hasn't fired before
//but with data the other way round
noValue (?meeting_b live:cdg-has-fired)

->

print("create distributed gathering has fired"),

(?dist_gath rdf:type meeting:Distributed-Gathering),

//create appropriate start time
(?dist_gath support:has-time-interval ?time),
(?time rdf:type support:Time-Interval),

//I want the start time of the most recent meeting
(?time support:begins-at-time-point ?most_recent),

//add the meetings as local events of the distributed gathering
(?dist_gath meeting:has-local-event ?meeting_a),
(?dist_gath meeting:has-local-event ?meeting_b),

(?meeting_a live:cdg-has-fired "fired")

]
```

10 - Create Additional Meeting

```
[createAdditionalMeeting:

(?a rdf:type live:Person-Present),

//this rule should only fire when there
//already is at least on meeting in session
(?existing_meeting rdf:type portal:Meeting-Taking-Place)
(?existing_meeting support:has-time-interval ?existing_time),
noValue(?existing_time support:ends-at-time-point),

//this location should be an ibutton reader
(?a portal:has-location ?loc),

//3store will have already been queried when getLocations fired
(?loc location:is-located-in ?room),
(?room rdf:type location:Meeting-Room),

//check that no meeting is currently in session in the room
noMeetingAtPhysLoc(?room),

//check that the person present event is not
//already in a meeting, this is in case it is part
//of a meeting that has finished
eventNotInMeeting(?a),

//get time interval and start time
(?a support:has-time-interval ?time_a),
(?time_a support:begins-at-time-point ?begin_time),

//get 'main agent'
(?a portal:has-main-agent ?person),

makeResource(?meeting),

makeResource(?time),

//check that this rule hasn't fired before on the same
//person present event
noValue(?a live:cam-has-fired)

->
```

10 continued:

```
print("createAdditionalMeeting has fired"),

//create a Meeting-Taking-Place containing the Person-Present
(?meeting rdf:type portal:Meeting-Taking-Place),

//create appropriate start time
(?meeting support:has-time-interval ?time),
(?time rdf:type support:Time-Interval),

(?time support:begins-at-time-point ?begin_time),

//also need to add room location
(?meeting portal:has-location ?room),

//assert a unique triple so we know this rule has already fired
(?a live:cam-has-fired "fired")

]
```

11 - Add Meeting To Distributed Gathering

```
[addMeetingToDistGath:

(?meeting rdf:type portal:Meeting-Taking-Place),

//match on a distributed gathering
(?dist_gath rdf:type meeting:Distributed-Gathering),

//check that the meeting is not already
//part of a distributed gathering
eventNotInDistGath(?meeting)

->

print("addMeetingToDistGath has fired"),

//add the meeting to the distributed gathering
(?dist_gath meeting:has-local-event ?meeting)

]
```

12 - Handle Sign Out

```
[handleSignOut:

//match on an iButton-Signed-In that has an end time
(?sign_in rdf:type live:iButton-Signed-In),
(?sign_in support:has-time-interval ?time_int),
(?time_int support:ends-at-time-point ?end_time),

(?sign_in live:id-of-ibutton-used ?id),

//ibutton will have already been resolved to a uri
//and resolved to a person when the corresponding sign_in ocurred
(?person live:has-personal-identifier ?ibutton),
(?ibutton live:has-ibutton-id ?id),

//get the corresponding Person-Present event
(?pp_event rdf:type live:Person-Present),
(?pp_event portal:has-main-agent ?person),
(?pp_event support:has-time-interval ?pp_time),

//check that the PP event is still active
noValue(?pp_time support:ends-at-time-point),

//check this rule hasn't already fired on this sign out
noValue(?sign_in live:hso-has-fired)

->

print("handleSignOut has fired"),

//assert the end time on the person present event
(?pp_time support:ends-at-time-point ?end_time),

publishToDataspace("TUPLE_TYPE", "TUPLE_EVENT"),
publishToDataspace("EVENT_TYPE", live:Person-Present),

publishToDataspace("ADD_TRIPLE", ?pp_time, support:ends-at-time-point,
?end_time),

publishToDataspace("PUBLISH"),
```

12 continued:

```
//finally delete the tuple for the start of the event
publishToDataspace("DELETE", ?pp_event),



//ensure that this rule only fires once per sign out
//by asserting a triple that says this rule has fired
(?sign_in live:hso-has-fired "fired")

]
```

13 - End Meeting During Distributed Gathering

```
[endMeetingDuringDistGath:

(?pp_event rdf:type live:Person-Present),
(?pp_event support:has-time-interval ?pp_timeint),
(?pp_timeint support:ends-at-time-point ?pp_end_time),

(?pp_event portal:has-location ?loc),
(?loc location:is-located-in ?room),
(?room rdf:type location:Meeting-Room),

//need this to match as otherwise rule will fire before all of
//the sign out event has been added
(?pp_end_time, support:year-of, ?year),
(?pp_end_time, support:month-of, ?month),
(?pp_end_time, support:day-of, ?day),
(?pp_end_time, support:hour-of, ?hour),
(?pp_end_time, support:minute-of, ?minute),
(?pp_end_time, support:second-of, ?second),
(?pp_end_time, meeting:millisecond-of, ?milli),

//check that the pp event is the most recent one - as that is
//the end time we need
eventHasMostRecentEndTime(?pp_end_time, live:Person-Present, ?room),

//check that the distributed gathering is still in session
(?dist_gath rdf:type meeting:Distributed-Gathering),
(?dist_gath support:has-time-interval ?timeint),
noValue(?timeint support:ends-at-time-point),

//match when a meeting is in session at the location
```

13 continued:

```
(?meeting rdf:type portal:Meeting-Taking-Place),
(?meeting portal:has-location ?room),
(?meeting support:has-time-interval ?meet_timeint),
noValue(?meet_timeint support:ends-at-time-point),

//ensure that pp_event is from this current meeting and not an
//earlier finished one that was in the same location
(?meeting portal:has-sub-event ?pp_event),

//see if there are no more participants at the meeting
//number must be quoted
participantsPresent(?meeting,"<=", "0"),

->

print("endMeetingDuringDistGath has fired"),

//assert an end time on the meeting
(?meet_timeint support:ends-at-time-point ?pp_end_time)

]
```

14 - End Distributed Gathering

```
[endDistGath:

(?meeting_finished rdf:type portal:Meeting-Taking-Place),
(?meeting_finished support:has-time-interval ?meet_timeint),
(?meet_timeint support:ends-at-time-point ?end_time),

//check that the meeting_finished is the most recent one - as
//that is the end time we need
eventHasMostRecentEndTime(?end_time, portal:Meeting-Taking-Place),

(?dist_gath rdf:type meeting:Distributed-Gathering),
(?dist_gath support:has-time-interval ?timeint),
noValue(?timeint support:ends-at-time-point),
(?dist_gath meeting:has-local-event ?meeting_finished),

onlyOneMeetingInSession(),

//match on the details of the final active meeting
```

14 continued:

```
(?meeting_active rdf:type portal:Meeting-Taking-Place),
(?meeting_active support:has-time-interval ?meet_active_timeint),
noValue(?meet_active_timeint support:ends-at-time-point),

//check that the final active meeting has more than one
//participant
//otherwise we need the special case to fire
//participantsPresent(?meeting_active, ">" ,"1"),

->

print("endDistGath has fired"),

(?timeint support:ends-at-time-point ?end_time)

]
```

15 - End Meeting After Distributed Gathering

```
[endMeetingAfterDistGath:

(?pp_event rdf:type live:Person-Present),
(?pp_event support:has-time-interval ?pp_timeint),
(?pp_timeint support:ends-at-time-point ?pp_end_time),

(?pp_event portal:has-location ?loc),

(?loc location:is-located-in ?room),
(?room rdf:type location:Meeting-Room),

//need this to match as otherwise rule will fire before all of
//the sign out event has been added
(?pp_end_time, support:year-of, ?year),
(?pp_end_time, support:month-of, ?month),
(?pp_end_time, support:day-of, ?day),
(?pp_end_time, support:hour-of, ?hour),
(?pp_end_time, support:minute-of, ?minute),
(?pp_end_time, support:second-of, ?second),
(?pp_end_time, meeting:millisecond-of, ?milli),

//check that the pp event is the most recent one - as that is
//the end time we need
```

15 continued:

```
eventHasMostRecentEndTime(?pp_end_time, live:Person-Present,?room),

//match on a meeting still in session
(?meeting rdf:type portal:Meeting-Taking-Place),
(?meeting support:has-time-interval ?meet_timeint),
noValue(?meet_timeint support:ends-at-time-point),

//ensure that pp_event is from this current meeting and not an
//earlier finished one that was in the same location
(?meeting portal:has-sub-event ?pp_event),

//check that there has been a distributed gathering and that it
//has ended
(?dist_gath rdf:type meeting:Distributed-Gathering),
(?dist_gath support:has-time-interval ?dist_gath_timeint),
(?dist_gath_timeint support:ends-at-time-point ?dist_gath_end_time),

//match when we are down to the last meeting participant
participantsPresent(?meeting,"<=","1")

->

print("endMeetingAfterDistGath has fired"),

//assert an end time on the meeting
(?meet_timeint support:ends-at-time-point ?pp_end_time)

]
```

16 - End Meeting Before Distributed Gathering

```
[endMeetingBeforeDistGath:

(?pp_event rdf:type live:Person-Present),
(?pp_event support:has-time-interval ?pp_timeint),
(?pp_timeint support:ends-at-time-point ?pp_end_time),

(?pp_event portal:has-location ?loc),

(?loc location:is-located-in ?room),
(?room rdf:type location:Meeting-Room),
```

16 continued:

```
//need this to match as otherwise rule will fire before all of
//the sign out event has been added
(?pp_end_time, support:year-of, ?year),
(?pp_end_time, support:month-of, ?month),
(?pp_end_time, support:day-of, ?day),
(?pp_end_time, support:hour-of, ?hour),
(?pp_end_time, support:minute-of, ?minute),
(?pp_end_time, support:second-of, ?second),
(?pp_end_time, meeting:millisecond-of, ?milli),

//check that the pp event is the most recent one - as that is
//the end time we need
eventHasMostRecentEndTime(?pp_end_time, live:Person-Present,?room),

//match on a meeting still in session
(?meeting rdf:type portal:Meeting-Taking-Place),
(?meeting support:has-time-interval ?meet_timeint),
noValue(?meet_timeint support:ends-at-time-point),

//ensure that pp_event is from this current meeting and not an
//earlier finished one that was in the same location
(?meeting portal:has-sub-event ?pp_event),

//check that there has not yet been a distributed gathering
noValue3(?dist_gath rdf:type meeting:Distributed-Gathering),

//match when we are down to the last meeting participant
participantsPresent(?meeting,"<=","1"),

->

print("endMeetingBeforeDistGath has fired"),

//assert an end time on the meeting
(?meet_timeint support:ends-at-time-point ?pp_end_time)

]
```

17 - Create Verbal Comment In Meeting

```
[createVerbalCommentInMeeting:

 //match on an active Microphone-Active event
 (?mic_act rdf:type live:Microphone-Active),
 //loc is a microphone-position
 (?mic_act portal:has-location ?loc),
 (?mic_act support:has-time-interval ?timeint),
noValue(?timeint support:ends-at-time-point),

 //get the room location of the
 (?loc location:is-located-in ?room),
 (?room rdf:type location:Meeting-Room),

 //get the ibutton reader located in the same seating position
 //as the microphone
 (?loc location:is-located-in ?seat_pos),
 (?seat_pos rdf:type live:Seating-Position),
 (?ibut_read_pos location:is-located-in ?seat_pos),
 (?ibut_read_pos rdf:type live:iButton-Reader-Position),

 //get the person seating in that seating position
 (?pp_event rdf:type live:Person-Present),
 (?pp_event portal:has-location ?ibut_read_pos),
 (?pp_event portal:has-main-agent ?person),
 (?pp_event support:has-time-interval ?pp_timeint),
noValue(?pp_timeint support:ends-at-time-point),

 //get the meeting that the pp_event is part of
 (?meeting portal:has-sub-event ?pp_event)

//create a resource to become the verbal comment
makeResource(?vc)

->

print("createVerbalCommentInMeeting has fired")

(?vc rdf:type meeting:Making-a-Verbal-Comment),
(?vc support:has-time-interval ?timeint),
(?vc portal:sender-of-information ?person),
(?vc portal:has-location ?room),
```

17 continued:

```
//set this verbal comment as a subevent of the meeting its part of
(?meeting portal:has-sub-event ?vc),

publishToDataspace("TUPLE_TYPE", "TUPLE"),
publishToDataspace("EVENT_TYPE", meeting:Making-a-Verbal-Comment),

publishToDataspace("ADD_TRIPLE", ?vc, rdf:type, meeting:Making-a-
Verbal-Comment),
publishToDataspace("ADD_TRIPLE", ?vc, portal:sender-of-information,
?person),
publishToDataspace("ADD_TRIPLE", ?vc, support:has-time-interval,
?timeint),

publishToDataspace("PUBLISH")

]
```

18 - Create Verbal Comment Outside Meeting

```
[createVerbalCommentOutsideMeeting:

//match on an active Microphone-Active event
(?mic_act rdf:type live:Microphone-Active),
//loc is a microphone-position
(?mic_act portal:has-location ?loc),
(?mic_act support:has-time-interval ?timeint),
noValue(?timeint support:ends-at-time-point),

//get the room location of the
(?loc location:is-located-in ?room),
(?room rdf:type location:Meeting-Room),

//get the ibutton reader located in the same seating position as
//the microphone
(?loc location:is-located-in ?seat_pos),
(?seat_pos rdf:type live:Seating-Position),
(?ibut_read_pos location:is-located-in ?seat_pos),
(?ibut_read_pos rdf:type live:iButton-Reader-Position),

//get the person sitting in that seating position
(?pp_event rdf:type live:Person-Present),
```

18 continued:

```
(?pp_event portal:has-location ?ibut_read_pos),
(?pp_event portal:has-main-agent ?person),
(?pp_event support:has-time-interval ?pp_timeint),
noValue(?pp_timeint support:ends-at-time-point),

//check that there is no meeting in session
noMeetingAtPhysLoc(?room),

//create a resource to become the verbal comment
makeResource(?vc)

->

print("createVerbalCommentOutsideMeeting has fired"),

(?vc rdf:type meeting:Making-a-Verbal-Comment),
(?vc support:has-time-interval ?timeint),
(?vc portal:sender-of-information ?person),
(?vc portal:has-location ?room),

publishToDataspace("TUPLE_TYPE", "TUPLE"),
publishToDataspace("EVENT_TYPE", meeting:Making-a-Verbal-Comment),

publishToDataspace("ADD_TRIPLE", ?vc, rdf:type, meeting:Making-a-
Verbal-Comment),
publishToDataspace("ADD_TRIPLE", ?vc, portal:sender-of-information,
?person),
publishToDataspace("ADD_TRIPLE", ?vc, support:has-time-interval,
?timeint),

publishToDataspace("PUBLISH")

]
```

19 - Handle Microphone Active End

```
[handleMicrophoneActiveEnd:

(?mic_act rdf:type live:Microphone-Active),
(?mic_act support:has-time-interval ?timeint),
(?timeint support:ends-at-time-point ?end),
```

19 continued:

```
(?vc rdf:type meeting:Making-a-Verbal-Comment),
(?vc support:has-time-interval ?timeint),

->

print("handleMicrophoneActiveEnd has fired"),

(?timeint support:ends-at-time-point ?end)

publishToDataspace("TUPLE_TYPE", "TUPLE_EVENT"),
publishToDataspace("EVENT_TYPE", meeting:Making-a-Verbal-Comment),

publishToDataspace("ADD_TRIPLE", ?timeint, support:ends-at-time-point,
?end),

publishToDataspace("PUBLISH"),

//finally delete the tuple for the start of the event
publishToDataspace("DELETE", ?vc)

]
```

20 - Archive Session

```
[archiveSession:

(?pp_event rdf:type live:Person-Present),
(?pp_event support:has-time-interval ?pp_timeint),
(?pp_timeint support:ends-at-time-point ?pp_end_time),
(?pp_event portal:has-location ?loc),
(?loc location:is-located-in ?room),
(?room rdf:type location:Meeting-Room),

noMeetingAtPhysLoc(?room),

->

print("archiveSession has fired"),
archiveSession()
]
```

9 References

[Acc04] The Access Grid,
http://www.accessgrid.org/, 2004. (verified 31st January 2005)

[Ack87] S. Acker and S. Levitt, "Designing videoconference facilities for improved eye contact", Journal of Broadcasting & Electronic Media, 31(2), pp181-191, 1987.

[AKT04] The AKT Reference Ontology, http://www.aktors.org/publications/ontology/, 2002. (verified 31st January 2005)

[Ala03] Harith Alani, Srinandan Dasmahapatra, Kieron O'Hara, and Nigel Shadbolt, "Identifying communities of practice through ontology network analysis", IEEE Intelligent Systems 18(2), pp. 18–25, 2003.

[Ama04] The Amaya Web Editor/Browser,
http://www.w3.org/Amaya/, 2004. (verified 31st January 2005)

[Ann04] The Annozilla plug-in,
http://annozilla.mozdev.org/, 2004. (verified 31st January 2005)

[Bac04] Bachler, M., Buckingham Shum, S., Chen-Burger, J., Dalton, J., De Roure, D., Eisenstadt, M., Komzak, J., Michaelides, D., Page, K., Potter, S., Shadbolt, N., Tate, A., "Collaborative Tools in the Semantic Grid", GGF11 - The Eleventh Global Grid Forum, Honolulu, Hawaii, USA, June 6-10, 2004.

[Bea01] Richard Beales, Don Cruickshank, David De Roure, Nick Gibbins, Ben Juby, Danius Michaelides and Kevin Page, "The Pipeline of Enrichment: Supporting Link Creation for Continuous Metadata", In The Proceedings of the 7th International Workshop on Open Hypermedia Systems, pp. 47-58, 2001.

[Bec01] Dave Beckett, "N-Triples", 2001,
http://www.w3.org/2001/sw/RDFCore/ntriples/ (verified 31st January 2005)

[Bec04] Dave Beckett (editor), "RDF/XML Syntax Specification (Revised)", W3C Recommendation, 10 February 2004

[Ber98] T.Berners-Lee, R. Fielding and L. Masinter. "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[Ber01]Tim Berners-Lee, James Hendler, Ora Lassila, "The Semantic Web", Scientific American, May 2001.

[Ber04] Tim Berners-Lee, "Notation3", July 2004, http://www.w3.org/DesignIssues/Notation3.html (verified 31st January 2005)

[Bet00] M. Bett, R. Gross, H. Yu, X. Zhu, Y. Pan, J. Yang and A. Waibel. "Multimodal Meeting Tracker". In Proc. RIAO2000 (Recherche d'Information Assistée par Ordinateur), Paris, France, April 2000.

[Bla98] S. Blake, D. Black, M. Carlson, E. Davies, E. Davies, Z. Wang and W. Weiss. "An Architecture for Differentiated Services", RFC 2475, December 1998.

[Bly93] S. A. Bly, S. R. Harrison, S. Irwin, "Media Spaces: bringing people together in a video, audio, and computing environment", Communications of the ACM, 36(1), pp. 28-46, 1993.

[Bra94] R. Braden, D. Clark and S. Shenker. "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994.

[Bri04] Dan Brickley, R.V. Guha, eds. "RDF Vocabulary Description Language 1.0: RDF Schema", W3C Recommendation, 10 February 2004.

[Bri05] Dan Brickley and Libby Miller, "The FOAF Vocabulary Specification", 3rd June 2005, http://xmlns.com/foaf/0.1/ (verified 20th June 2005)

[Bux97] W. Buxton, A. Sellen, M. Sheasby, “Interfaces for Multiparty Videoconferences”, In K.E. Finn, A.J. Sellen, & S.B. Wilbur (Eds), Video-mediated communication, pp. 385-400, 1997, New Jersey: Lawrence Erbaum Associates.

[Car89] Nicholas Carriero and David Gelernter, “Linda in Context”, Communications of The ACM, 32(4), pp. 444-458, April 1989.

[Chi03] Patrick Chiu, Qiong Liu, John Boreczky, Jonathan Foote, Tohru Fuse, Don Kimber, Surapong Lertsithichai, and Chunyuan Liao “Manipulating and annotating slides in a multi-display environment”, In Proc. INTERACT '03, pp. 583-590, September 1, 2003.

[Chi99a] P. Chiu, A. Kapuskar, S. Reitmeier, and L. Wilcox. “Meeting Capture in a Media Enriched Conference Room”. In Proceedings of the Second International Workshop on Cooperative Buildings (CoBuild'99), Lecture Notes in Computer Science, Vol. 1670 Springer-Verlag, pp. 79-88, 1999.

[Chi99b] P. Chiu, A. Kapuskar, S. Reitmeier and L. Wilcox. “NoteLook: Taking Notes in Meetings with Digital Video and Ink”, In Proc. 7th ACM Conference on Multimedia, 1999.

[Cir04] Fabio Ciravegna, Sam Chapman, Alexiei Dingli and Yorick Wilks, “Learning to Harvest Information for the Semantic Web”, in Proc. 1st European Semantic Web Symposium, Heraklion, Greece, May 10-12, 2004.

[Cor91] D. Corkhill, “Blackboard Systems”, AI Expert, 6(9), pp 40-47, September 1991.

[Cru01] Don Cruickshank, Luc Moreau, David De Roure, “Architectural Design of a Multi-Agent System for Handling Metadata Streams”, In Proc. Fifth International Conference on Autonomous Agents, pp 505-512, 2001.

[Cut02] Ross Cutler, Yong Rui, Anoop Gupta, JJ Cadiz, Ivan Tashev, Li-wei He, Alex Colburn, Zhengyou Zhang, Zicheng Liu, Steve Silverberg. “Distributed Meetings: A

Meeting Capture and Broadcasting System", In Proc. 10th ACM Conference on Multimedia, 2002.

[Dea04] M. Dean, G. Schreiber, eds. "OWL Web Ontology Language Reference", W3C Recommendation, 10 Feb 2004.

[Den04] Laurent Denoue, Gurminder Singh, Arijit Das, "Taking Notes on PDAs with Shared Text Input", In Proc. ED-Media 2004, June 21, 2004.

[DeR04]De Roure, D. and Hendler, J.A., "E-Science: the Grid and the Semantic Web", IEEE Intelligent Systems, 19(1), pp 65-71, 2004.

[Dou92] P. Dourish and S. Bly, "Portholes: Supporting Awareness in a Distributed Work Group", Proceedings of the Conference on Human Factors in Computing Systems, Monterey, CA, pp 541-547, 1992.

[eBi04] eBiquity: RGB Ontologies, http://ebiquity.umbc.edu/v2.1/ontology/, 2004.

[Eng62] Douglas C. Engelbart. "Augmenting Human Intellect: A Conceptual Framework". Summary Report AFOSR-3223 under Contract AF 49(638)-1024, SRI Project 3578 for Air Force Office of Scientific Research, Stanford Research Institute, Menlo Park, Ca., October 1962.

[Eng75] Douglas C. Engelbart. "NLS Teleconferencing Features: The Journal, and Shared-Screen Telephoning". In Proceedings of the COMPCON Conference, pp. 173-176, 1975.

[EQU04] The EQUATOR Interdisciplinary Research Collaboration, http://www.equator.ac.uk, 2004. (verified 31st January 2005)

[Equ04b] Equip4j,
http://www.crg.cs.nott.ac.uk/~cmg/Equator/, 2004 (verified 31st January 2005)

[Flo97] Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", IEEE/ACM Transactions on Networking, 5(6), pp. 784-803, December 1997.

[FOA05a] "The FOAF-a-Matic", http://www.ldodds.com/foaf/foaf-a-matic.html (verified 20th June 2005)

[FOA05b] "FOAF Explorer", http://xml.mfd-consult.dk/foaf/explorer/ (verified 20th June 2005)

[For82] C. L. Forgy, "RETE: A Fast Algorithm for the Many Patterns/Many Objects Match Problem", Artificial Intelligence, 19(1), pp. 17-37, 1982.

[Gol03] Jennifer Golbeck, Bijan Parsia, and James Hendler, "Trust Networks on the Semantic Web", In Proc. Cooperative Intelligent Agents 2003, Helsinki, Finland, August 2003.

[Goo86] G. O. Goodman, M. J. Abel, "Collaboration research in SCL", In Proc. The 1986 ACM Conference on Computer-Supported Cooperative Work, Austin Texas, pp 246-251, Dec 3-6, 1986.

[Gre00] Chris Greenhalgh, Jim Purbrick, Dave Snowdon, "Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring", In Proc. Third International Conference on Collaborative Virtual Environments, pp 119-127, San Francisco, California, 2000.

[Gre02] Chris Greenhalgh, "EQUIP: An Extensible Platform For Distributed Collaboration", In Proc. WACE 2002, Edinburgh, UK, 2002.

[Gro03] Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh, eds. "XPointer Framework", W3C Recommendation, 25 March 2003.

[Gut96] C. Gutwin, S. Greenberg, "Workspace Awareness For Groupware", In Proc. CHI '96, pp 208-209, 1996.

[Gut97] C. Gutwin "Workspace Awareness in Real-Time Distributed Groupware", Ph.D. Thesis, University of Calgary, Canada, 1997.

[Har03] Harris, Stephen and Gibbins, Nicholas, "3store: Efficient Bulk RDF Storage", In Proc. 1st International Workshop on Practical and Scalable Semantic Web Systems, Sanibel Island, Florida, USA. pp1-15, 2003.

[Hil81] S. R. Hiltz and M. Turoff, "The evolution of user behavior in a computerized conferencing system", Communications of the ACM, 24(11), pp. 739-751, 1981.

[Holl92] Jim Hollan and Scott Stornetta. "Beyond Being There". In Proc. ACM CHI'92, pp. 119–125, 1992.

[Hua03] K. Huang, M. M. Trivedi, "Video arrays for real-time tracking of person, head, and face in an intelligent room," Machine Vision and Applications, 14(2), pp. 103-111, June 2003.

[iBu04] iButton - Contact Memory, http://www.ibutton.com/, 2004. (verified 31st January 2005)

[Jav04], JavaSpaces, http://java.sun.com/developer/products/jini/index.jsp, 2004. (verified 31st January 2005)

[Jen04] The Jena Semantic Web Framework, http://jena.sourceforge.net/, 2004. (verified 31st January 2005)

[Jub03] Benjamin Juby and David De Roure, "Real-Time Speaker Identification and Participant Tracking in The Access Grid", In Proc. 4th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PG Net 2003) , pp. 313-319, Liverpool, UK, June 2003.

[Kah01] José Kahan, Marja-Riitta Koivunen, Eric Prud'Hommeaux, and Ralph R. Swick, "Annotea: An Open RDF Infrastructure for Shared Web Annotations", in Proc. of the WWW10 International Conference, Hong Kong, May 2001.

[Koi03] Marja-Riitta Koivunen, Ralph Swick, Eric Prud'hommeaux, "Annotea Shared Bookmarks", In Proc. of the KCAP 2003 workshop on knowledge markup & semantic annotation, Sanibel, Florida, October 2003.

[Kot04] Alan Kotok, Ralph Swick, "The Zakim IRC Teleconference Agent", http://www.w3.org/2001/12/zakim-irc-bot.html, 2004. (verified 31st January 2005)

[Man97] A. Mané, "Group space: The role of video in multipoint videoconferencing and its implication for design", In K.E. Finn, A.J. Sellen, & S.B. Wilbur (Eds), Video-mediated communication, pp. 401-414, 1997, New Jersey: Lawrence Erbaum Associates.

[Mar99] W. Mark."Turning pervasive computing into mediated spaces", IBM Systems Journal, 38(4), pp 677-692, 1999.

[Mem05] The Memetic Project, http://www.memetic-vre.net/ (viewed 25th June 2005)

[Mik00] Mikic I., Kohsia H., Trivedi M., "Activity monitoring and summarization for an intelligent meeting room" Proceedings IEEE Workshop on Human Motion, Austin Texas, December 2000.

[Mil68] R. B. Miller, "Response time in man-computer conversational transactions", In Proc. AFIPS Fall Joint Computer Conference, Vol. 33, pp 267-277, 1968.

[Mil92] D. Mills "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, March 1992.

[Mil04] I.C. Millard, D.C. De Roure, N.R. Shadbolt, "The use of ontologies in contextually aware environments", In Proc. First International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham, UK, pp.42-47, 2004

[Min93] Scott. L. Minneman, Steve. R Harrison, "Where Were We: making and using near-synchronous, pre-narritive video", In Proceedings of ACM Multimedia '93, pp. 207-214, 1993.

[Min04] MINDSWAP Conference Ontology, http://www.mindswap.org/~golbeck/web/www04photo.owl, 2004. (verified 31st January 2005)

[MYS04] The MySQL Database Server, http://www.mysql.com/, 2004. (verified 31st January 2005)

[Pag01] Kevin Page, Ben Juby, Richard Beales and David De Roure, "Continuous Metadata". In Proc. 2nd Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking & Broadcasting (PGNET 2001), pp.265-269, Liverpool, UK, 2001.

[Pay02a] Payne, T. R. and Miller, L., "Calendars, Schedules and the Semantic Web". ECRIM News(51), pp. 16-17, 2002.

[Pay02b] Payne, T. R., Singh, R. and Sycara, K., "Calendar Agents on the Semantic Web", IEEE Intelligent Systems 17(3), pp. 84-86, 2002.

[Ram04] S. D. Ramchurn, B. Deitch, M. K. Thompson, D. C. De Roure, N. R. Jennings, and M. Luck, "Minimising Intrusiveness in Pervasive Computing Environments using Multi-Agent Negotiation", In Proc. First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04), pp. 364-372, Boston, Massachusetts, 2004.

[RDQ03] Hewlett-Packard Labs, RDQL - RDF data query language, http://www.hpl.hp.com/semweb/rdql.htm, 2003. (verified 31st January 2005)

[Rod91] T. A. Rodden. "A Survey of CSCW Systems", Interacting with Computers, 3(3), pp 319-353, 1991.

[Seg00] Bill Segall, David Arnold, Julian Boot, Michael Henderson and Ted Phelps "Content Based Routing with Elvin4", In Proc. AUUG2K, Canberra, Australia, June 2000.

[Sel92] Sellen, A., Buxton, W. & Arnott, J. "Using spatial cues to improve videoconferencing", in Proc. CHI '92, pp 651-652, 1992.

[Sch03] Ronald Schroeter, Jane Hunter, Douglas Kosovic, "Vannotea – A Collaborative Video Indexing, Annotation and Discussion System For Broadband Networks" In Proc. KCAP 2003 workshop on knowledge markup & semantic annotation, Sanibel, Florida, October 2003.

[sch04] schraefel, m. c., Shadbolt, N. R., Gibbins, N., Glaser, H. and Harris, S. "CS AKTive Space: Representing Computer Science in the Semantic Web," In Proc. 2004 World Wide Web Conf., ACM Press, 2004.

[Ste86] M. Stefik, D. G. Bobrow, S. Lanning, D. Tatar, G. Foster, "WYSIWIS revised: early experiences with multi-user interfaces", In Proc. The 1986 ACM Conference on Computer-Supported Cooperative Work, Austin Texas, pp 276-290, Dec 3-6, 1986.

[Ste87] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning and L. Suchman, "Beyond The Chalkboard: Computer support for collaboration and problem solving in meetings", Communications of the ACM, 30(1), pp. 32-47, 1987.

[Swi04] Ralph Swick, "The RRSAgent IRC Bot Description", http://www.w3.org/2002/03/RRSAgent, 2004. (verified 31st January 2005)

[TSp04] TSpaces – Intelligent Connectionware, http://www.almaden.ibm.com/cs/TSpaces/, 2004. (verified 31st January 2005)

[Tur02] O. Turk, O. Sayli, H. Dutagaci, L. Arslan, "A Sound Source Classification System Based on Subband Processing", In Proc. 7th International Conference on Spoken Language Processing, September 2002.

[Ver97] R. Vertegaal, "Conversational Awareness in Multiparty VMC", Extended Abstracts of ACM CHI'97 Conference on Human Factors in Computing Systems, Atlanta, GA, 1997.

[Ver98] R. Vertegaal , H. Vons , R. Slagter, “Look Who’s Talking: The GAZE Groupware System”, In Proc. CHI '98, pp 293-294, April 1998.

[Wai03] Alex Waibel, Tanja Schultz, Michael Bett, Mathias Denecke, Robert Malkin, Ivica Rogina, Rainer Stiefelhagen, Jie Yang “SmaRT: The Smart Meeting Room Task at ISL”, In Proc. IEEE International Conference on Accoustics, Speech and Signal Processing, pp. 752-755, Hong Kong, April 2003.

[Yam96] Yamaashi, K., Cooperstock, J., Narine, T. & Buxton, W. “Beating the limitations of camera-monitor mediated telepresence with extra eyes”. In Proceeding of CHI '96, pp 50-57, 1996.