Architectural Design of a Multi-Agent System for Handling Metadata Streams

Don Cruickshank Luc Moreau David De Roure

Department of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ UK
{dgc,L.Moreau,dder}@ecs.soton.ac.uk

ABSTRACT

We have designed a multi-agent architecture to deliver metadata streams synchronously with multimedia streams over a wide-area network. To this end, we have devised a simple protocol for synchronising agents to a media clock. This protocol defines the concept of a deadline, after which servers can drop data because it can no longer reach clients in time. We also introduce a new concept of a contract as a first-class entity representing a successful subscription; a contract is used by agents as a session identifier during the navigation of streams. Quality of service is a vital element of this architecture because of the need to deliver metadata on time. As a result, our architecture supports various communication protocols, including UDP, RMI, SSL, or multicast. This resulted in a return to a more declarative form of speech acts, totally orthogonal to a notion of virtual communication channel used to manage the quality of service of communication.

1. INTRODUCTION

The international television programme 'Big Brother' takes place in a specially built house that contains many cameras and microphones. Contestants are placed in the house and are periodically evicted by a voting system that involves the house members and the viewing public. Besides television updates, the house is viewable by a fixed number of continuous video streams across the Internet; each stream is individually produced, carrying salient content from the available cameras and microphones, and is delayed by a few seconds to enable some basic editing.

This scenario involves multiple live multimedia data streams, together with extra information added in real time by the production activities. The positions of the cameras and of the contestants provide additional data, needed to make sense of the streams, and the status of the voting process might also be conveyed. Commentary may make reference to earlier scenes and related resources. Subtitling can also be provided. All this information *about* the multimedia streams is *metadata*; metadata enables the understanding of other data, and itself must have a common structure to be understood. We note from this scenario that this metadata is continuous, is necessarily synchronised to the multimedia streams and can be carried in separate network flows rather than embedded in the multimedia data. Using the metadata, we can repurpose the multimedia streams for multiple applications; for example, the end user can customise their viewing. The user's actions may themselves result in additional metadata.

The automation of the metadata production and processing is a distributed problem which yields naturally to the agent paradigm. We have conceived a distributed architecture, based on clients with which users may interact and servers responsible for delivering streamed media and metadata streams. We have adopted the agent paradigm to build such an architecture because it provides us with the flexibility required by our target applications: *(i)* We envision metadata to be generated by a vast number of distributed competing agents [17]. *(ii)* We expect clients to adapt their behaviour according to the user's needs; the use of agents (and user models) has been advocated for such interfacing tasks [14]. *(iii)* Clients are expected to be proactive and may initiate the downloading of new metadata streams, and process metadata on the fly (such as filtering, merging, etc.). *(iv)* Finally, high-level negotiation between clients and server may be used for dealing with payment or negotiating access to servers.

The processing of live metadata is not a traditional application of software agents, and it raises a number of challenges. Broadly we need to map some aspects of multimedia and metadata transmission into the agent communication framework, effectively using the agent communication language to provide the functionality of a control channel and to convey metadata. Although the agent communication layer is not intended to carry multimedia data per se, individual agents might process this data. The purpose of this paper is to describe the new concepts that we introduce in our multi-agent system in order to deal with these streams of metadata; they can be summarised as follows:

1. The design of a synchronisation protocol and associated ontology allowing clients to synchronise metadata servers with a media clock and servers to determine when real-time data need to be dropped.

2. A concept of a contract as a first-class embodiment of a successful subscription, and its use as a session identifier.

3. A communication infrastructure based on a concept of performatives orthogonal to a notion of startpoint/endpoint, used to manage the quality of services of communication.

4. A communication framework able to support various communication protocols, including UDP, RMI, SSL, or multicast.

These concepts are all being implemented in SoFAR (the SOuthampton Framework for Agent Research), a Java-based multi-agent framework [17].

This paper is organised as follows. In Section 2, we describe further applications requiring metadata streams. We use these to draw a set of requirements, which we present in Section 3. On the basis of these requirements, we have designed an agent infrastructure, first overviewed in Section 4; next, the salient contributions of our agent architecture are presented in Section 5. We then describe the implementation in the SoFAR agent framework in Section 6 and discuss our results in Section 7.

2. METADATA STREAMS

The following three scenarios further motivate the need for metadata streams and form the basis for our requirements:

1. A live news broadcast can be augmented with metadata carrying information about the news items. This could include catalogue information for news footage, information about rights to use the material, subtitles and links to associated resources such as online documents. Agents can use the metadata to filter new items or bring them to the user's attention, translate subtitles and perform appropriate searches and customisation.

2. A musician might transmit a digital audio stream from their instrument to an agent which provides a synchronous stream of MIDI data. Another agent could match the MIDI data against a database in order to provide accompaniment, related music or relevant hypermedia links. Note that in this example, one agent receives multimedia data and produces metadata. We have exercised this scenario using a search engine based on melodic pitch contours [2].

3. A teacher might record a presentation so that students can view it retrospectively, in which case the replay is live but there is elapsed time during which metadata can be created for subsequent streaming. Metadata includes references to other teaching resources, perhaps in the form of synchronised hypermedia links which the user's agent could choose to follow according to personal preferences.

The first scenario resembles the 'Big Brother' scenario but with greater diversity of sources; it emphasises use of both live material and library footage. In the second, an agent processes multimedia data to produce metadata which could in fact be regarded as a live multimedia stream in its own right; the scenario could extend to synchronous collaboration with other musicians and agents. In contrast, the metadata in the third scenario need not be live but could in principle be downloaded in advance of replay; however, there might still be metadata generated on-the-fly by the audience in the form of annotations and links.

3. REQUIREMENTS

In this section, we draw a series of requirements from the scenarios, which helped us design our new agent architecture. We focus on a network of agents that deliver metadata streams to multimedia clients. We have not attempted to design a system that transmits media streams using an agent communication language because protocols and solutions for streaming media across networks already exist, such as the Java Media Framework [13]. As the media is played to the user, the respective agents collaborate to get metadata transferred to the multimedia client, so that the client can mix the metadata in to produce a value-added media stream.

It is a requirement of our scenarios that our application must be able to run over a wide area network, such as the Internet, and we anticipate the provision of "quality of service" (QoS) support using the techniques now becoming established (such as resource reservation, or differentiated services). We have identified different quality of service needs: reliable communications (typically TCP based) will be assumed for non-streamed message exchanges between agents. On the subject of metadata streams, on-time delivery is preferred over reliable delivery; therefore, less reliable protocols such as UDP could be adopted for metadata streaming. The mechanism for transferring metadata should not be restricted by low grade backchannels: the streams may be transmitted from a satellite, and may require a lower bandwidth device, such as a modem connection, for the receiver to communicate with the sender. Furthermore, the delivery of metadata cannot be stalled by a requirement of user to service agent communication during normal operation, where a normal operation is defined as watching a channel without switching channels or without navigating (fast forward, rewind) within the channel.

In any case, whatever the protocol, we do not assume negligible communication latency, nor do we assume that latency in one direction of a particular route is equivalent to latency in the other direction. Each pair of agents that share a synchronised communication channel is made aware of the time difference between them. Should certain pieces of metadata take too much time to obtain, then it is better for the server to realise that they cannot be delivered on time, and to simply drop those pieces of metadata from the stream.

It is important to understand that we consider *live* scenarios in addition to pre-recorded programmes. In a live scenario, metadata on the media is being created at the same time as the media is being created, and is also viewed by the audience *all at the same time.* We are investigating a system where each of these can occur as part of a pipeline of processes; a viewer can watch the Big Brother house as a continuous programme, and receive metadata that relates to the programme whilst the media is still being filmed at the house. If we were to restrict ourselves to pre-recorded material, we would only need to transfer an archived block of metadata to the user agents at the start of transmission.

4. ARCHITECTURE OVERVIEW

In this section, we present the network architecture and terminology to suit our requirements of Section 3.

The overall architecture is shown in Figure 1. The *media server* is a network server that delivers a media stream to a *multimedia client*. The multimedia client displays the stream with annotations based on the stream of metadata received from the *metadata agent network*. We show that the media server is separate from the network of agents. The solid lines in the diagram denote *streams* of data, and the dotted lines denote agent matchmaking.

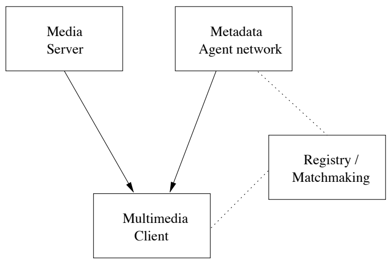

Figure 1: Architectural Overview

The user has control over the playback of the media source, so any navigation that the user invokes on the streamed media is also sent to the agent network. In a multicast environment, we assume that caching of the media stream occurs between the originating source and the multimedia client, so that navigation requests of one user do not interfere with other users.

In Figure 2, we investigate the components of the agent network. The metadata streams, shown as solid lines, are each associated with a producer and a consumer of metadata. In the context of metadata streams, we will refer to the producer of a metadata stream as a *service agent*. The consumer is referred to as a *user agent*. In the figure, we show that one of the service agents acts as a user agent for another service agent. Furthermore, the multimedia client is also a user agent that receives the metadata stream.

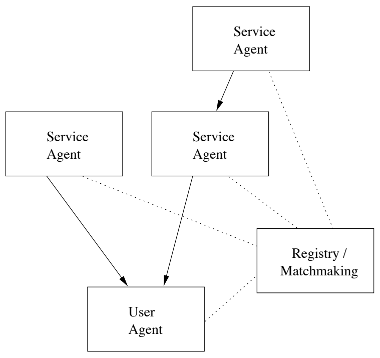

Figure 2: Agent Network

In the network, there are a number of network nodes that are separated from each other in terms of locality and latency. We rely on the notion of a *registry* to allow agents within the network to advertise services. Agents that require a service will ask the registry for a list of agents with the requested capability.

Communication between agents that reside on different network nodes may cross physical or organisational boundaries. An organisation that generates the metadata might deliver to an intermediary (e.g. an internet service provider) to add value to their service, rather than directly to end users.

5. MULTI-AGENT DESIGN

The behaviour of our architecture may be summarised as follows. Using a matchmaking mechanism, user agents discover and select metadata servers, from which they subscribe to metadata streams. Each successful subscription establishes a session between a user agent and a service agent. Within each session, the user agents have the ability to synchronise the metadata stream using a synchronisation protocol. In this section, we present the novel aspects of our architecture: *(i)* a new synchronisation protocol, *(ii)* a concept of a deadline for service agents to deliver metadata, *(iii)* a concept of a contract for session handling, *(iv)* a view of performatives orthogonal to the notion of virtual channel for handling quality of service. In Section 6, we will describe their concrete realisation in the SoFAR agent framework.

5.1 The Synchronisation Protocol

We define the *synchronisation protocol* as a sequence of actions by which a user agent that receives synchronised metadata can give the service agent some indication about the network latency from the service agent to itself. The synchronisation protocol that we describe does not depend on a global clock, but rather on the time difference between a known clock on the server and the media clock on the user agent.

We opted for *just in time* delivery, according to which the server sends data to the user just in time for it to be used. Delivering metadata just in time gives us two properties. Firstly, we reduce the storage overheads required by user agents, as user agents are required to pre-buffer less data. Secondly, user agents can expect meta-data to be more reactive; the period of time between an event occurring and the resulting metadata arriving at the user agent is reduced.

The difference between the known clock on the service agent and the clock on the user agent remains constant, because they progress at the same rate. Once the time difference is derived, the user agent can inform the service agent of that difference, and the server can then stream metadata to the user without the need for user to service agent communication.

The sequence of events that are used to determine the service to user agent latency is shown in Figure 3. Three timelines represent the *media clock*, the *user clock*, both on the user agent, and the *server clock*. The user clock and the server clock are synchronised with each other, because they are supposed to progress at the same rate. The media clock and the user clock may differ by a speed ratio, noted ρ, typically decided by the user; in our example the ratio is 2, indicating that the user is viewing the media at twice the speed.

The user agent sends a first message requesting the service agent to send back the server clock value, s_{stime}. As soon as the user agent receives the reply, it records its clock value as

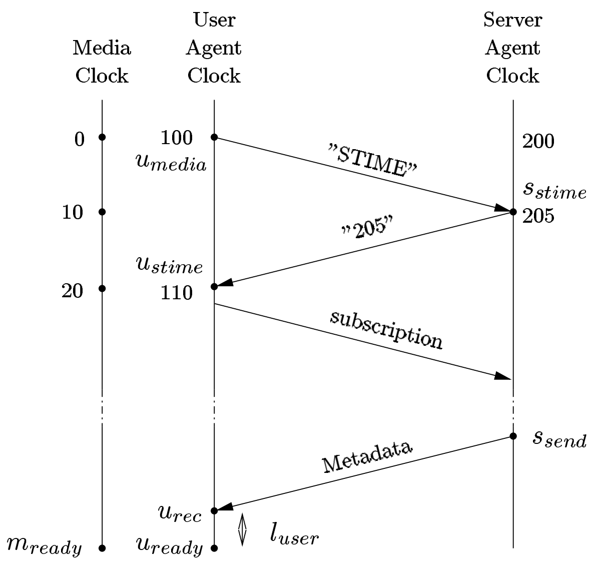

Figure 3: Synchronisation Protocol

u_{stime}. In Figure 4, we call α the difference between these two clocks, which takes into account the network latency between the service agent and the user agent.

$$\begin{aligned}
\alpha &= u_{stime} - s_{stime} && \text{(User-Server Clock Diff.)}\\
u_{rec} &= s_{send} + \alpha && \text{(User Receiving Time)}\\
u_{ready} &= u_{rec} + l_{user} && \text{(User Ready Time)}\\
m_{ready} &= (u_{ready} - u_{media}) \times \rho && \text{(Media Ready Time)}\\
\vec{\delta} &= (u_{media}, l_{user}, \alpha, \rho) && \text{(Synchronisation Tuple)}
\end{aligned}$$

Figure 4: Synchronisation Equations

Using this information, the service agent can now determine when a message is received by the user agent u_{rec} when sent by the server at time s_{send}. Besides the latency, the user agent may need some lead time (l_{user}) to allow further processing on metadata between their reception and the displaying of results to the user. We define u_{ready} as the time (on the user agent clock) by which data are ready, when sent by the service agent at s_{send}.

In itself, the user agent time is not very useful to the service agent: the service agent wishes to determine when it is suitable to send a piece of metadata related to a specific time on the media. The user agent knows about the media that it is receiving, so it can determine what the user clock value would be if the media clock is at zero (u_{media}). In Figure 3, u_{media} is 100 because the media clock is zero at that time on the user agent.

Provided the service agent receives the synchronisation quadruple u_{media}, l_{user}, α and ρ, named $\vec{\delta}$ in Figure 4, from the user agent, the service agent can now determine by what time on the media clock a message would be ready to be displayed by the user agent m_{ready}.

For example, in Figure 3, $\alpha = 110 - 205 = -95$. For a lead time $l_{user} = 10$, and a send time $s_{send} = 230$, we obtain $u_{ready} = 230 - 95 + 10 = 145$. And therefore the corresponding media time is $m_{ready} = (145 - 100) * 2 = 90$.

We have shown how the service agent can be informed of the time difference between itself and a user agent. In a busy network, the user agent might find that the network performance changes over a long period of time. In an implementation of this protocol, the user agent would need to regularly request the server time from the service agent to reflect changes in network latency. By communicating the synchronisation quadruple $\vec{\delta}$ the user agent will be able to keep the service agent synchronised to the media played.

5.2 Real-Time Aspect

There is a real-time dimension in our scenarios because metadata is expected to be delivered to clients synchronously with multimedia streams. Our notion of real-time is directly derived from the synchronisation protocol of Section 5.1.

Let us consider a piece of metadata which concerns a section of the media, represented by an interval $[m_{start}, m_{end}]$, expressed in media clock. Thus the service agent can determine if it is too late for sending a message containing this metadata to the user agent. It just needs to compare m_{ready} (obtained from the server send time s_{send}) with m_{end}.

$$late? = m_{ready} > m_{end}$$

If a message has become too late, then the service agent knows that it can be dropped. Alternatively, the service agent can use the same equations to schedule the work of other agents that produce metadata, so that metadata becomes available to the service agent, in time for delivery to the user agent.

5.3 Sessions and Contracts

A *subscription* is the action by which an agent declares its interest in receiving some message related to a specified topic. If a server answers positively to a subscription act, it commits itself to inform the subscribee about facts that match its interest. A server issues a *contract* as a result of the subscription act and as a proof of its commitment.

The role of a contract is threefold: *(i)* The contract is a first-class embodiment of a successful subscription, and a proof of the server's commitment to deliver data to a client; should the server fail to deliver, the contract could be used as evidence in a conflict resolution procedure. *(ii)* A successful subscription essentially creates a session between a client and server, and a contract also acts as a session identifier; the contract may be used by the client to control the session. *(iii)* If a client wishes to stop the arrival of a stream, it just has to terminate the associated contract. If contracts are shared and exchanged by agents, they are able to detect the termination, between distributed agents, of the subscription procedure; in other words, from the server's point of view, a subscription will terminate only when all associated contracts have been terminated.

In our agent architecture, we use subscription as the mechanism to initiate a metadata stream to a user agent. A session should be distinguished from a conversation, as defined in FIPA [5]. A session may be seen as a sub-conversation with a specific quality of service, possibly using a totally different communication protocol.

From a technical point of view, our idea of contract bears a strong resemblance to leases introduced in Java RMI [12] and popularised by Jini [18]. Our Java implementation of contracts is in fact based on leases; they are also useful for handling client crashes.

5.4 Message Intent vs. Message Transport

Subscription and contracts are the mechanism by which a client and a server agree to establish a session in which metadata is delivered to the client as a stream of messages. Metadata is streamed according to a quality of service, which has to be negotiated between the client and the server. For instance, metadata could be streamed either as UDP datagrams, or in a QoS-enabled network, or even on a secure socket layer. The key aspect is that the required quality of service is not known at design-time, but it will only be decided at run-time.

We believe it is important that an agent framework does not prescribe the means through which communicative acts actually take place, but instead it does specify their intention; this philosophy allows a declarative style for communications, which facilitates the reusability of components. Therefore, we have based communications on speech act theory [21]. We have identified a minimal set of *performatives* which represent the most common communication patterns, such as querying, informing, registering, subscribing or requesting. Other forms of communicative acts must be defined by composing the primitive ones.

In order to specify the required quality of service, we introduce a further mechanism, orthogonal to the concept of performative. A communication between two agents is based on a "virtual link" defined by a startpoint and an endpoint, which are concepts borrowed from the communication library Nexus [6]. An *endpoint* identifies an agent's ability to receive messages using a specific communication protocol; it extracts messages from the communication link and passes them on to the agent. A *startpoint* is the other end of the communication link, from which messages get sent to an endpoint. There may be several startpoints for a given agent, each acting as a representative (or proxy) of the agent at remote locations. Vice-versa, several endpoints may be associated with a startpoint, which allows some form of multicasting.

We shall note that we have adopted speech acts similarly to standard agent communication languages KQML [4] and FIPA[5]; however, we only define a minimal set of these, corresponding to the most frequent communication patterns. Therefore, we do not support domain specific performatives, such as forwarding messages, but instead we express them by using our basic performatives and a suitable communication ontology. In addition, any quality of service aspect for communication, such as digital signatures, encryption or even QoS negotiation and transport protocol selection is handled with the startpoint/endpoint mechanism.

6. IMPLEMENTATION

In this section, we describe the realisation of our agent architecture for handling metadata streams, and the extensions we introduced in SoFAR (the SOuthampton Framework for Agent Research) for that purpose.

6.1 Ontology

Here we define an ontology for metadata streams, their synchronisation and their navigation. In the synchronisation protocol, the service agent returns to the user agent the server clock value expressed as an absolute time; the associated relation appears in Figure 5.

A piece of metadata that relates to a certain period of time within an identifiable stream is represented by using the relationship described in Figure 5. *Temporal-metadata* is related to a multimedia stream, specified by its identifier, perhaps a URI, and an indication of when the metadata is relevant; the *start* and *end* times are given in terms of the media clock. In addition, the synchronisation quadruple $\vec{\delta}$, noted "Delta" in Figure 5, contains all the parameters required by the synchronisation protocol.

The complete *temporal-metadata* entity allows agents to exchange pieces of temporal metadata with respect to a deadline. A service agent that has been informed of a deadline, via a subscription or a navigation request, can then set a deadline for its own sources.

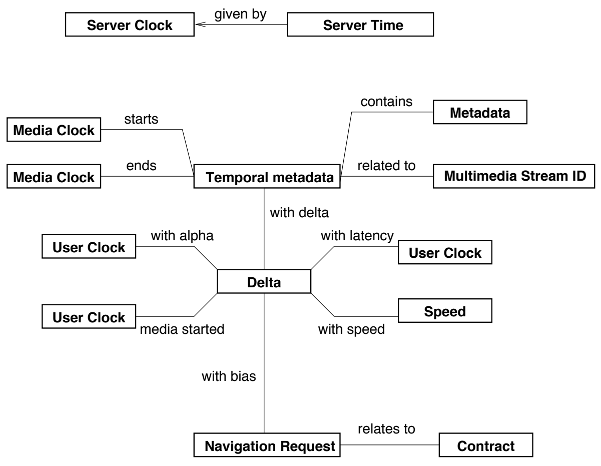

Figure 5: Ontology for temporal metadata

The user agent may navigate through the media stream by use of fast forwarding or rewinding the media stream. Actions like these require a recalculation of the $\vec{\delta}$ quadruple, and the service agent needs to be informed of its new value.

6.2 Communication Model Implementation

Practically, when an agent A_1 wishes to perform a speech act with another agent A_2, A_1 activates the corresponding method on a startpoint of A_2. The startpoint creates a communication context object, serialises the data, sends it to the endpoint, which deserialises the data and passes it to the agent A_2. When the speech act produces a result, it is returned back, via the endpoint, to the startpoint; it is then returned as the result of the method invocation on the startpoint. (Note that there is also an asynchronous variant of the performatives.)

SoFAR supports multiple communication mechanisms, all accessible through the startpoint/endpoint paradigm. Typically, an agent will have an endpoint for each communication protocol it wishes to support. Its peers will choose the associated startpoint according to the desired quality of service for their interaction. For the purpose of delivering metadata streams, we support Java Remote Method Invocation, secure socket layer based communications and UDP (based on the Ninja implementation of RMI [24]). Ninja also supports proper multicast which we wish to investigate along with QoS-enabled communications provided by IPv6.

6.3 Protocol Implementation

We now turn to the conversation between agents that synchronise and exchange temporal metadata. Using the FIPA convention for protocol representation [5], Figure 6 shows the communication of the server time request conversation from the user agent's perspective. The user agent may perform the server time request act at any time, and does not require a contract with the service agent beforehand.

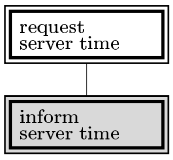

Figure 6: User Agent Server Time

The user agent may request the server time as often as it wishes. The user agent is free to draw its own conclusion on a suitable $\vec{\delta}$ quadruple for use with a particular agent, such as second guessing variance in network latency. To perform this, an agent might simply take an average from a number of samples, or perhaps predict variance in latency from previous experience.

It is important to note that the server must send its answer to a server time request via the same kind of network route that the metadata will take. An implication of this is that over a UDP channel, the server time result might not be received by the user agent. Components of the underlying network are liable to drop UDP packets under heavy load conditions. In this case, the user agent may decide that after a certain period of time, the answer is deemed to be lost. If the service agent does not reply to a number of server time requests, it might decide that the quality of the respective metadata stream will behave similarly, and look for another service agent.

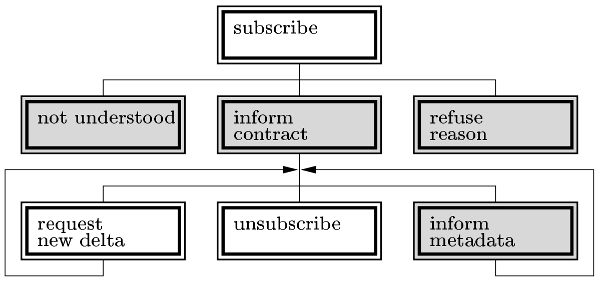

Figure 7: User Agent Transitions

Figure 7 shows the exchange of messages during a subscription from the perspective of the user agent. The user agent initiates the conversation by requesting a subscription to the service agent. The subscription is in the form of a *temporal-metadata* entity, which contains only a multimedia stream ID and a $\vec{\delta}$ quadruple. Upon receipt of a contract, the user agent is ready to receive *temporal-metadata* entities that match the original subscription, but with the remaining fields filled in by the service agent.

The user agent is capable of two acts: a navigation request or to break the contract. If the media that is played to the user has changed in play speed, then the user agent requests the service agent to accept a new $\vec{\delta}$ quadruple. The effect of this action to the incoming metadata stream can only occur as quickly as the round-trip time between the two agents, thus any metadata that is received during this period may not be useful to the user agent. The subscription is terminated by breaking the contract with the service agent. Similarly with the navigation request, metadata may still arrive for a short period after the breaking of the contract.

The round-trip delay in metadata stream control, in conjunction with unexpected variance in network latency, means that the user agent can receive metadata that is already late or perhaps too early (rewinding, for example) to sensibly use. Thus a realistic user agent implementation will filter out incoming metadata that is not usable for its purpose.

If the user agent, during normal play conditions, realises that a significant proportion of metadata is late, then it is able to adjust the $\vec{\delta}$ quadruple accordingly.

The server agent's perspective of a subscription is shown in Figure 8. Once a subscription is accepted and the contract is received by the user agent, the service agent can send metadata from other sources, or from a local cache of metadata.

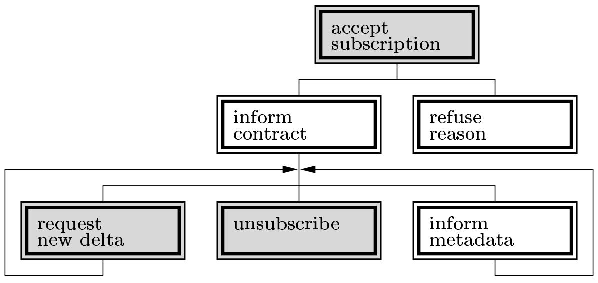

Figure 8: Service Agent Transitions

The architecture we have adopted in our implementation uses the notion of a hypermedia link service [3], which resolves the source anchor of a link to all the possible destinations by querying a link database ('linkbase') to identify relevant links. Links may be regarded as metadata. In our implementation, the service agent requested metadata from a linkbase and stored the results in the local cache of the service agent. For a subscription, the following occurs. At regular intervals, the cache is examined, and any metadata that is close to the *late*? predicate is sent to the user agent. Metadata that is already late is still added to the cache, in case the user agent rewinds the metadata stream.

Our experience of the protocol showed us that both the user and service agent are capable of making intelligent decisions on metadata delivery, as both are able to determine the lateness of entities in the metadata stream. In addition to video streams, our implementation works equally well with audio streams.

7. FUTURE AND RELATED WORK

Metadata streaming is a new application domain for multi-agent systems, and there is relatively little related work. Multi-agent systems have previously been applied to the configuration and management of networks, for example for

ATM networks in [8, 10, 15, 19]. In [11], agents are used in determining the spanning tree for multicast routing; this takes advantage of mobility, which we have not addressed in this paper but is a subject of current work within the SoFAR framework. With respect to quality of service, Hashimoto et al [9] propose a 'flexible multimedia system' for multimedia teleconferencing, applying negotiation protocols. The support for negotiation and collaboration in multi-agent systems is one of our motivations for adopting this paradigm for metadata streams but is beyond the scope of this paper.

The synchronisation protocol that we propose helps service agents to determine when metadata is too late and need to be dropped and when to schedule the search or the creation of metadata. Research on how agents can schedule their work in order to meet real-time deadlines [7, 22] could be used to define service agents.

With respect to streams and signalling associated with synchronisation, there are two relevant standards that have influenced our thinking. Network Time Protocol (NTP) [16] provides a more comprehensive solution to clock synchronisation and discusses issues of skew and drift between clocks. Real Time Streaming Protocol [20] provides a framework for controlled, on-demand delivery of real-time data. Like our system, RTSP does not carry the multimedia data itself, but rather the signalling information needed to provide 'network remote control'. These messages can also be mapped into our performatives, and this in fact forms the basis of a demonstrator system which is currently under development.

The actual metadata conveyed in our system is opaque but in practice will typically be in XML format; in interoperating with existing systems we are adopting the Resource Description Format (RDF) developed by the World Wide Web Consortium [23].

Having designed and built this infrastructure for delivering metadata over wide-area networks, a number of fundamental research issues have to be addressed, and we discuss some of them in the rest of this section.

Scope of Service Discovery - Advertisement. In an wide area network, we need mechanisms to control the scope of discovery and advertisement of capabilities. We consider that the agent framework might extend across the Internet, comprising of thousands of nodes separated by varying network latencies. We cannot expect a user agent to search every node for potential servers, nor can we expect a service agent to advertise a service to every node. Either of these would result in a solution that would not scale.

We are investigating techniques such as hierarchical registry organisation and the use of Jini [18] to provide a scalable and controllable mechanism by which resources may be advertised and found.

Service agents may also want to control which user agent accesses a metadata stream; multi-agent negotiation [1] could be a means to provide the required level of control. Furthermore, agents would possibly wish to preserve the authenticity if not the privacy of the transmitted metadata; digital signatures and encryption techniques may be used for that purpose.

Multiple Clients. A single service agent may deliver data streams to several user agents. In particular, a single multicast stream could be sent to several user agents. In such a case, it becomes difficult for individual user agents to control the synchronisation. Clever caching strategies may have to be introduced for that purpose, in particular when clients have a limited caching capability.

Mobility. We are also considering the possibility of user and service agents migrating from node to node. If agents move about, then the route between them changes in term of physical locations, and possibly network latency times. We need to be able to renegotiate the synchronisation between the agents. We would certainly like to consider a scenario where a service agent tries to move closer to multiple user agents by using only local knowledge of the network.

8. CONCLUSION

In this paper, we have applied the multi-agent paradigm to a new type of application over the Internet, namely the delivery of metadata streams. From the requirement of this application, we have devised a synchronisation protocol and associated ontology, which allow user agents to navigate and synchronise metadata streams. Our investigation in the quality of service has led us to conceive a communication architecture, where the communication intent expressed as a performative is completely orthogonal to the notion of virtual communication link, used to specify the required quality of service. We have implemented our prototype into the SoFAR agent framework, which supports multiple types of communication protocols.

9. ACKNOWLEDGMENTS

This research is supported by EPSRC project "HyStream" reference GR/M84077/01. The authors are grateful to Kevin Page of the IAM Research Group for insightful discussions about continuous metadata, and to colleagues at BT Laboratories for their collaboration in this research project.

10. REFERENCES

[1] M. Beer, M. d'Inverno, M. Luck, N. R. Jennings, C. Preist, and M. Schroeder. Negotiations in multi-agent systems. *Knowledge Engineering Review*, 14(3):285–289, 1999.

[2] Steven G. Blackburn and David C. De Roure. A tool for content based navigation of music. In *Proceedings of ACM Multimedia '98*, pages 361–368, September 1998. ISBN: 1-58113-036-8.

[3] L.A. Carr, D.C. De Roure, H.C. Davis, and W. Hall. Implementing an open link service for the world wide web. *World Wide Web Journal*, 1(2), 1998.

[4] T. Finin, Y. Labrou, and J. Mayfield. *Software Agents, J. Bradshaw, Ed.*, chapter KQML as an Agent Communication Language. MIT Press, 1997.

[5] FIPA: Foundation for Intelligent Physical Agents. `http://drogo.cselt.stet.it/fipa/`.

[6] Ian Foster, Carl Kesselman, and Steven Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1996.

[7] Alan Garvey and Victor Lesser. A Survey of Research in Deliberative Real-Time Artificial Intelligence. Technical report, Unviersity of Massachusetts, 1993.

[8] M. A. Gibney and N. R. Jennings. Dynamic resource allocation by market-based routing in telecommunications networks. In *2nd Int. Workshop on Multi-Agent Systems and Telecommunications*, pages 102–117, 1998.

[9] Koji Hashimoto, Yoshitaka Shibata, and Norio Shiratori. Agent-orented flexible multimedia system considering organization and qos functions. In *Proceedings of the Tenth International Workshop on Database and Expert Systems Applications*, pages 662–666, Florence, Italy, September 1998. IEEE Computer Society.

[10] Alex L. G. Hayzelden, John Bingham, and Zhiyan Luo. A multiagent approach for distributed broadband network management. In *Proceedings of the Fourth International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 179–192, London, UK, April 1999.

[11] Stefan G. Hild and Jorg H. Bischof. Agent-based multicast routing. In *Proceedings of the Third International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 57–81, London, UK, March 1998.

[12] *Java Remote Method Invocation Specification*, November 1996.

[13] Java Media Framework API. `http://java.sun.com/products/java-media/jmf/`, 1995.

[14] Pattie Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[15] M. Miller, D. Krieger, N. Hardy, C. Hibbert, and E. Tribble. An Automated Auction in ATM Network Bandwidth. In *Market-Based Control. A Paradigm for Distributed Resource Allocation*, pages 96–125. World Scientific, 1996.

[16] David L. Mills. Network time protocol (version 3) specification, implementation. Request for Comments RFC1305, Network Working Group, March 1992.

[17] Luc Moreau, Nick Gibbins, David De Roure, Samhaa El-Beltagy, Wendy Hall, Gareth Hughes, Dan Joyce, Sanghee Kim, Danius Michaelides, Dave Millard, Sigi Reich, Robert Tansley, and Mark Weal. SoFAR with DIM Agents: An Agent Framework for Distributed Information Management. In *The Fifth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, pages 369–388, Manchester, UK, April 2000.

[18] Scott Oaks and Henry Wong. *Jini In a Nutshell.* O'Reilly, 2000.

[19] Jide Odubiyi, George Meekins, Song Huang, and Tracy Yin. Proteus*–adaptive polling system for proactive management of ATM networks using collaborative intelligent agents. In *Proceedings of the third annual conference on Autonomous Agents*, pages 402–403, 1999.

[20] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). Request for Comments RFC2326, Network Working Group, April 1998.

[21] John Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, 1969.

[22] Ignacio Soto, Mercedes Garijo, Carlos A. Iglesias, and Manuel Ramos. An Agent Architecture to fulfill Real-Time Requirements. In *Autonomous Agents 2000*, Barcelona, Spain, 2000.

[23] W3C. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation REC-rdf-syntax-19990222, World Wide Web Consortium, February 1999.

[24] Matt Welsh. Ninja. A scalable Internet services architecture. http://www.cs.berkeley.edu/˜mdw/proj/ninja/, 1999.