Streaming Support for Multimodal Data in Smart Spaces

Yanhua Mao  
Department of Computer Science, Tsinghua University, Beijing, 100084, P.R. China  
+86-13691428487  
maoyanhua@tsinghua.org.cn

Runting Shi  
Department of Computer Science, Tsinghua University, Beijing, 100084, P.R. China  
+86-13161820091  
shirunting@mails.tsinghua.edu.cn

WeiKai Xie  
Department of Computer Science, Tsinghua University, Beijing, 100084, P.R. China  
+86-13801212505  
xwk@263.net

Yuanchun Shi  
Department of Computer Science, Tsinghua University, Beijing, 100084, P.R. China  
+86-13601239798  
shiyc@tsinghua.edu.cn

Guangyou Xu  
Department of Computer Science, Tsinghua University, Beijing, 100084, P.R. China  
+86-10-62784141  
xgy-dcs@mail.tsinghua.edu.cn

ABSTRACT

The idea of ubiquitous computing and smart spaces is no longer a dream and has long been an area of serious research. Various smart space projects have been built in research institutions all around the world. The Software Infrastructure of Smart Space (SISS), as an integral part of these projects, is to coordinate and manage numbers of software and hardware modules in the environment. The limitation of existent SSIS's, however, is that they seldom provide streaming support for potential multimodal and multimedia applications transferring real-time data in Smart Space.

In the Smart Classroom Project at the Multimedia Lab of Tsinghua University, we integrated into our SISS, the Smart Platform, a stream-oriented communication scheme based on Real-time Transport Protocol (RTP) and IP multicast. It is a common solution that caters for the varied needs of several categories of streaming applications in a Smart Space. Relevant design issues will be detailed in this paper.

Keywords

Smart Platform, stream-oriented communication, Smart Space.

1. INTRODUCTION

Since the year 2000, our research group has been devoted to the development of a tele-education oriented “Smart Classroom”[1][2], a Smart Space integrated with many perceptual and multimodal technologies such as speech recognition and machine vision, so as to give the teacher and the remote students a better user experience.

The Smart Platform[3], a multi-agent system, is developed as the supporting software infrastructure for the Smart Classroom. For the coordination of the software and hardware modules in the environment, the platform exploits a message-oriented communication scheme based on a subscribe-publish model. Due to its relatively poor scalability and prolonged transmission delay, this style of communication is not suitable for some categories of applications, especially those that require the real-time streaming of high volumes of data.

As a complement to the message-oriented communication, streaming support has been added to the platform to provide transport service to a number of real-time applications that are sensitive to the delivery latency and jitter, e.g. a user tracking module constantly reporting the location of the user to other modules in environment.

In designing the functionalities of the streaming service at the software infrastructural level, we need to take into account the diversified needs of different applications. Application specific issues include the choice between the push and pull mode, and whether verification should be provided to check the integrity of data. Our implementation opens several options to application developers so that they could tailor the basic service to their own needs.

Finally, according to the fundamental principles of SSIS’s, this stream-oriented communication should be kept lightweight, i.e. it should introduce minimal system overhead and require little effort in understanding on the application developers’ side.

The rest of the paper is structured as follows: In Section 2, we introduce different categories of communication in a Smart Space, and demonstrate the motivation of our work with interesting scenarios where stream-oriented communication could be applied. After presenting the overall design of the streaming service in Section 3, we explore the issue of intra-stream synchronization in Section 4. Section 5 and 6 discuss application specific and implementation issues respectively whereas Section 7 presents an evaluation of the stream-oriented communication. Finally, following the related works section, the conclusion and future work are given.

2. MOTIVATION

2.1 Categories of Communication in a Smart Space

Two basic categories of communication can be seen in a Smart Space, namely the message-oriented and the stream-oriented.

The former type of communication is typically exploited for the delivery of sporadic messages involving a limited amount of data. These messages usually have high-level semantics, are sensitive to
2.2.1 The Laser Pointer Tracker

These scenarios are to be used as basic examples throughout the paper. Classroom project that constitute the motivation of our work in providing streaming support at the software infrastructural level. We shall now present two interesting scenarios of the Smart application that send a notification to a text-to-speech module instructing it to pronounce a word, or when a computer vision module reports that a new person has come into the environment.

By contrast, the stream-oriented communication is employed when live video/audio or some constantly recurring events are transferred between modules. Typically these applications would involve real-time data with a low semantic level; while they impose strict temporal constraints on the transmission service, they usually have loose requirements on the integrity of data. In additions, these applications sometimes involve multi-point communication where several parties need to share data originated from a single.

With the rise of multimedia technologies, a greater variety of real-time multimedia applications are being integrated into Smart Spaces. Many of them pose the same challenge, i.e. the streaming of high volumes of time-sensitive data, therefore driving the need for a common streaming service at the software infrastructural level. Though existing SISS’s have well addressed the task of the message-oriented communication that facilitates the coordination of the software and hardware modules, they provide poor support for real-time streaming, typically by establishing dedicated UDP paths between the parties concerned with no further transport layer enhancement service.

2.2 Some Scenarios

We shall now present two interesting scenarios of the Smart Classroom project that constitute the motivation of our work in providing streaming support at the software infrastructural level. These scenarios are to be used as basic examples throughout the paper.

2.2.1 The Laser Pointer Tracker

In a conventional classroom, a hand-held laser pointer is typically used by a teacher to highlight the focus of his/her slides or blackboard writings. Through a computer vision module named Laser2Cursor that tracks the position of the laser spot, the Smart Classroom takes advantage of the laser pointer as a tool of human-computer interaction, so that the teacher is entitled to a conventional classroom experience he has been accustomed to. For instance, as the arbitrator of the floor, a desk-top based tele-education system will require the teacher to input his decision with a mouse or keyboard. In the Smart Classroom, however, the physical presence of the computer is removed from the sight of the teacher who simply aims the laser pointer at a remote student’s representing image to give him the floor or revoke it (See Figure 1). In addition, the teacher could use the laser pointer to operate on his slide presentation, e.g. switching between the pages.

The internal algorithm of the Laser2Cursor module is beyond our concern. But it is the duty of the Smart Platform, the underlying software infrastructure of the Smart Classroom, to provide high-quality delivery service so that the shifting focus of the laser pointer can be transmitted to relevant modules in a smooth and timely fashion.

2.2.2 The Signal Splitter

Another interesting scenario arises when the speech of the teacher needs to be delivered to several modules simultaneously, for instance, to a module named SameView that delivers the live audio feed to remote students, and meanwhile, to a Speech2Command module where speech recognition and semantic analysis are performed to translate the speech into commands which the computer will automatically carry out for the teacher, facilitating him/her with his/her lecture. These two modules may reside in separate hosts, but they need to utilize the same audio stream. While a possible solution is to use a hardware splitter to split the audio stream into two, this scheme is inconvenient as it involves manual configuration; it is not scalable as a hardware splitter has a fixed number of outputs; and it is inflexible when the stream has to be routed to dynamic or ad hoc targets.

An alternative scheme is to deliver the audio stream via network to all subscribed modules. A paradigmatic approach within this scope would be one that shows good consideration for the characteristics of multimedia streaming while keeping the network overhead at a reasonable level.

2.3 The Task

Both scenarios introduced above involve applications that show real-time streaming characteristics. Though multimedia streaming has been much studied over the past few decades, to readily employ it in a Smart Space, much remains to be done in its integration into the underlying software infrastructure.

The stream-oriented communication scheme, to be incorporated into the communication layer of the Smart Platform, need to have the following characteristics: (1) It should ensure intra-stream synchronization [4] which will be detailed in section 4. (2) It should support the “one-to-many” mode of communication that makes possible the concept of the “software splitter”. (3) Application specific issues should be taken into account so that application developers may customize the service according to individual needs. (4) The streaming service must be seamlessly integrated into the existing SISS. (5) Performance issues should also be taken into consideration: the computation power and network resources must be utilized in an efficient and economic fashion.
3. The Basic Design
The overall design of the streaming service is illustrated in Figure 2.

![Figure 2. The Basic Structure of the Streaming Service](image)

As intra-stream synchronization is required by many real-time streaming applications, the Smart Platform ought to include this functionality as part of the streaming service.

4. Intra-stream synchronization
We shall begin this section by explaining the concept of intra-stream synchronization and its necessity, followed by a concrete approach involving several initial buffering policies whose strengths and weaknesses will be examined.

4.1 What is Intra-stream synchronization
Generally speaking, intra-stream synchronization is the preservation of temporal signature in a single stream [6], such that the same temporal relations (i.e. temporal order and intervals) between basic units of data should be observed at the destination the way data were produced at the source. In multimedia applications, intra-stream synchronization could even out the jitter caused by network transmission to achieve the smooth playback of audio/video data.

4.2 Design Issues
Our solution to intra-stream synchronization incorporates the commonly adopted timestamping scheme to represent the temporal relations between data units, and a receiver buffering pool to smooth out delivery jitter. See Figure 3.

![Figure 3. Timestamping and receiver buffering are combined to achieve intra-stream synchronization](image)

Assume the source of the stream generates data units d1, d2, d3, d4 at time t, t+Δt1, t+Δt2, t+Δt3, t+Δt4 respectively. The payload will be timestamped so that at the receiving side the units will be submitted to the application in relative order at time T, T+Δt5, T+Δt6, T+Δt7, T+Δt8, where T+Δt is the latency incurred by network transmission and end system buffering. And in case d2 is lost or damaged during transmission, d1, d3, d4 shall yet be presented to the receiving application at T, T+Δt3, T+Δt4, T+Δt6, T+Δt7, T+Δt8 respectively.

Though the basic idea is straightforward, several issues need to be dealt with: (1) corruption or loss of data may occur during transmission, whose consequences are to be understood; (2) a policy is needed to adjust the initial buffering length for the receiver buffer; (3) measures have to be taken to handle possible buffer underflows.

Because most real-time streaming applications can tolerate a mild degree of packet losses although the variation in delivery latency may prove detrimental, we simply adopt the best-effort delivery service provided by the network layer without offering any error recovery enhancement. While transport layer services including error recovery, flow control and congestion control are far more complicated for IP multicast than unicast and may introduce new problems regarding scalability and system overhead, our plain solution based on RTP/IP multicast would suffice in most situations, and meanwhile keep the streaming service lightweight.

To smooth out transmission jitter, a certain number of data packets have to be buffered before playback. As long as the receiver buffer does not fall short of data, the total latency is a constant value which is easier for the application to handle than variable latencies. Take the Laser Pointer Tracker as an example, users have found it much easier to accustom themselves to a constant system reaction time than one that is unpredictable.

The initial buffering length is defined as the number of data units buffered before the first unit is submitted to the application layer.
The initial buffering length is directly associated with the latency between the streaming source and the destination. Whereas real-time interactive applications are more sensitive to delay, other applications may prefer a longer initial buffering in exchange for a better chance to cope with out-of-order packets or bursty traffic resulting from unstable network conditions. Different initial buffering policies will be examined in Section 4.3.

The receiver buffer suffers from an underflow in case the playback rate has outpaced the data supply rate, eventually draining the receiver buffer. This may happen when the source has temporarily cut off provision, or when the network experiences a congestion, resulting in prolonged transmission delay or loss of data. Whenever a buffer underflow occurs, the playback will be withheld until a certain number of units, as specified by the Initial Buffering Length, have refilled the buffer.

A brief description of the buffering strategy is shown in Figure 4. The waiting state is where the receiver playback is suppressed and initial buffering is performed, and the active state, where playback is in operation.

![Figure 4. A two-state Finite State Machine that describes the receiver buffering strategy](image)

### 4.3 Initial Buffering Policies

An Initial Buffering Policy is to determine the Initial Buffering Length, a threshold value that will trigger the transition of the FSM from the waiting state to the active state (See Figure 4). We propose several alternative strategies implemented as options in the streaming service of the Smart Platform.

#### 4.3.1 Buffering for a Fixed Number of Units

The most straightforward approach is to directly specify the number of data units to buffer before initial playback. That is to say, the FSM in Figure 4 will shift from the waiting state to the active state, when the following condition is met:

$$\text{Count(buffer)} \geq N,$$

where $N$ is a user specified parameter.

In practice, though, this strategy may not turn out so simple as it is in notion, for users may soon find it difficult to come up with a good estimate of $N$. For one thing, the relationship between the initially buffered data units as measured in count and the total latency as measured in time is not really straightforward: it depends on the actual frame rate of the streaming source, and may vary with the dynamic network conditions. It would be preferable, therefore, if a measurement could be provided in the dimension of time.

#### 4.3.2 Buffering for a Fixed Amount of Time

In this policy, the receiver buffers for a fixed amount of time before initial playback starts. The condition for the corresponding state transition is represented formally as below:

$$\text{MaxTimeStamp(buffer)} - \text{MinTimeStamp(buffer)} \geq T,$$

where $T$ is a user specified parameter.

When the network transmission delay is negligible in comparison with the initial buffering time, $T$ gives a reasonable estimate of the total latency which is hardly predictable in the previous scheme. Application developers may find this policy easier to apply especially in real-time interactive applications where inferences can be drawn from a user experience study on the maximum perceived latency acceptable. In the Laser Pointer Tracker application, for instance, experience shows that a system reaction time smaller than 500ms is well taken by the users. Here $T$ is specified to be 400ms, so that when summed up with the time consumed by computation and network transmission, it will ensure a total reaction time below 500ms.

Though this policy allows the user to predict the total latency regardless of the data supply rate and network dynamics, undesirable situations may arise under poor network conditions, for when data loss is severe, playback may start with relatively few data units buffered. The buffer may soon be emptied, causing the FSM (See Figure 4) to fall back to the waiting state where playback is checked and initial buffering starts again. The overall effect is the repeated transitions between the two states and the intermittent playback as perceived by application users.

#### 4.3.3 The Hybrid Policy

By combining the previous two schemes, the hybrid policy came into being. Again we provide a formal expression of the condition:

$$\{\text{MaxTimeStamp(buffer)} - \text{MinTimeStamp(buffer)} > T\} \text{ and } \{\text{Count(buffer)} > N\}$$

where both $T$ and $N$ are user specified parameters which, if carefully chosen, may allow the user to leverage the strengths of both former policies.

### 5. Application Specific Issue

This section discusses application specific issues including whether the push or pull mode should be employed and whether data consistency check is needed.

#### 5.1 Pull vs. Push

For the receiving application, the incoming stream data could be processed in either a pull mode where the application actively pulls the receiver buffer for incoming data, or a push mode where the platform’s streaming service invokes a callback function in the receiver application with one or more data units whenever they are ready for playback. In the former case, it is left to the application to ensure intra-stream synchronization, and in the latter, the duty rests with the platform.

With some applications, intra-stream synchronization is implemented as an embedded part, for instance, an available video/audio decoder with a complicated internal algorithm to handle synchronization. In such cases, a simple pull mode will suffice, and the application is left with a greater degree of freedom in achieving its own scheme of intra-stream synchronization.

Other applications, however, may prefer the SISS to perform the cumbersome task of intra-stream synchronization for them. Such is the case with the Laser Pointer Tracker whose developer would have toiled to implement his own buffer pool hadn’t intra-stream synchronization been provided as a ready service.
In the streaming service of the Smart Platform, the two modes are provided as options so that application developers may apply whichever he finds useful.

5.2 Verification of Data Integrity
When network errors arise, data may be corrupted during transmission and rendered useless. In Section 3.2 we briefly discussed the consequences of data damage and argued about the need to include an error recovery mechanism at the SISS layer.

Yet even when no error recovery mechanism is present, it may be desirable, in some situations, for the SISS to check the consistency of incoming data before they are handed over the application layer. Needs may vary with applications though, for instance, a video streaming application must be more resilient to errors than the Laser Pointer Tracker where an erroneous position of the laser spot may result in an unexpected operation.

In the streaming service provided by the Smart Platform, an optional CRC64 checksum may be applied to the payload, so that corrupted data units will be discarded upon detection.

6. Binding with the Smart Platform
In this section we describe the integration of the streaming service into our SISS, the Smart Platform.

6.1 The Group Based Communication
We shall begin with an overview of the basic concepts of the Smart Platform.

An Agent is an individual functional module that works as the basic unit of encapsulation in the Smart Platform. The Directory Service (DS) is a central service provider responsible for the registration of agents, the dispatching of messages and system management.

The communication layer of the Smart Platform works on a uniform conceptual model, namely the group based communication model. A message group designates a service or an aggregate of services, and in the message-oriented communication mechanism, any message published to a particular group will be automatically relayed by the DS to subscriber agents.

The notion of a group also applies in the stream-oriented communication mechanism. Yet the DS no longer serves as an intermediate point in dispatching the messages, it simply assigns a unique multicast address, which will be made known to every subscriber and publisher of the group. In fact a streaming group can be regarded as an access point where the sender and receivers directly exchange data. The Smart Platform adopts the Single Source Multicast (SSM) model where only one source is allowed for each group. While the “many to many” mode of communication can be achieved through the allocation of multiple groups, i.e. one for each sender, the SSM model is simpler to manage, and presents a clear-defined conceptual model to application developers.

For each streaming group, the DS maintains a metadata description encoded in XML where various properties of the stream are specified. Here XML is employed as the encoding scheme to exploit its features of extensibility, interoperability and readability. The information, supplied by the publisher of the group, is open for query to all agents. Meanwhile, the DS will notify the subscribers once the information is updated by the streaming source. Unlike the streaming data, the metadata is transferred through the message-oriented communication service so as to ensure the reliability of its transmission. Another important feature is that publisher and subscribers may join the streaming group in arbitrary order. This is to guarantee the loose coupling between agents and to enhance the robustness of the system against partial failures. Figure 5 illustrates the architecture of the group-based streaming service.

![Figure 5. The source agent register the streaming service with the DS. Receiver agents may query the DS for the multicast address of the group, so that a streaming path will be established directly between the sources and subscribers.](image)

6.2 Agent API
Some Agent APIs are provided by the Agent Development Library for agent developers to access the stream-oriented communication service of the Smart Platform. The basic elements are listed as below:

JoinAsSource is used to register a stream source with the Directory Service. Basic properties of the stream must be specified, including the maximum packet size, and whether to turn on the data consistency check. The source may also provide extended properties for the stream, e.g. media content or format specification. Upon successful return, the source will be able to call PublishStream to send stream samples.

SubscribeStream must be called before the stream subscriber can start receiving. Upon successful return, the subscriber will be instructed to listen on a specific multicast address where stream data and metadata will hence be received. When this primitive is called, the user must specify a callback method through which stream data, whenever they are ready, will be passed on to the application layer.

Finally LeaveAsSource and UnsubscribeStream are called when a streaming source or subscriber is to leave the group.

The Agent API is available in C++ for the Windows platform. Because interoperable is featured in our basic design of the Smart Platform, it won’t be hard to migrate it onto heterogeneous platforms.

7. Experiment
Experiments have been conducted to evaluate the performance of the stream-oriented communication of the Smart Platform. The first experiment is intended to give a numerical profile of performance metrics. As the quality of user experience is much emphasized notion in a Smart Space, a second experiment, in which the “software splitter” was realized for a video stream, was...
carried out to gather user feedback on the perceived quality of the video data.

In the first experiment, the Smart Classroom was used as the testbed. Four Pentium IV 1.4GHz PCs were involved; each having 128MB RAM and Windows 2000 Professional installed as the operating system. All these PCs were connected via a dedicated 100Mbps Ethernet. Four source agents, each residing in a separate PC, were programmed to generate fixed-length samples at a constant rate. Meanwhile, on each PC were 4 receiver agents subscribing to the 4 sources respectively. In all, 20 agents were run on 4 hosts and 4 stream-groups or RTP sessions, each having 1 source and 4 subscribers, were established. Metrics including the damage/loss rate and the standard deviation of the delivery latency are studied and listed in Table 1.

### Table 1. Standard deviation of the delivery latency (ms) /damage(loss) rate

<table>
<thead>
<tr>
<th>sample length</th>
<th>40</th>
<th>400</th>
<th>4000</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample interval (ms)</td>
<td>100</td>
<td>50</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3.21 / 0%</td>
<td>3.37 / 0%</td>
<td>3.87 / 0%</td>
</tr>
</tbody>
</table>

With the intra-stream synchronization mechanism enabled, the variance in delivery latency mainly results from the limited precision of the operating system timer when the stream timing service is still capable to process the incoming data. Table 1 demonstrates that in the current implementation, the variance is acceptable when sampling is performed every 20ms, yet the timing service is not capable of handling accuracy of 10ms or below. Meanwhile, the damage/loss rates are extremely low. This is because the maximum throughput in the experiment is around 1.6MB/s (4000bytes*100*4), only about 16% of the total capacity of the network. Finally, during the experiment, the observed CPU load of the PCs remains low (less than 5%), which accounts for another desirable feature of the streaming service, i.e. a minimal system overhead.

### 8. Related Work

NIST’s Smart Flow System [7] provides an architecture to support data flows in Smart Spaces. IP multicast is also used in Smart Flow System MetaGlu[e][8] is the software infrastructure of MIT’s Intelligent Room Project[9]. It glues together large groups of software agents and a wide range of communication and discovery services. MetaGlu is implemented in Java, replacing the remote method invocation (RMI) mechanism. It focuses more on the coordination of modules and addresses less on the streaming support in Smart Spaces

The D-Stampede [10] system, from the College of Computing, Georgia Institute of Technology, focuses on programming support for the distributed heterogeneous computing elements in a ubiquitous computing environment. It addresses the interactive, dynamic, and stream-oriented nature of potential multimedia applications and develops appropriate computational abstractions...
in the D-Stampede distributed programming system. This project is more like a distributed computing environment than a Smart Space environment.

9. Conclusion and Future Work
In this paper, we presented the design of the stream-oriented communication scheme provided as a fundamental service of the Smart Platform. Based on RTP over IP multicast, our streaming service seeks to solve the common challenge posed by a class of real-time multimedia applications that need to stream multimodal data in a Smart Space. Meanwhile, it takes application specific issues into account and provides several optional features so that users could tune the basic streaming service for their particular needs.

Whereas intra-stream synchronization has already been realized, we shall set out to explore the necessity and potentiality of supporting inter-stream synchronization[4] as well. Meanwhile, the streaming service will be utilized by many HCI models in the Smart Classroom.

10. ACKNOWLEDGMENTS
Thanks to those who have contributed to the Smart Platform Project, especially to the design and implementation of the stream-oriented communication. Changhao Jiang, who graduated from our lab as a Master of Engineering, designed the Smart Platform and conceived a rudimentary picture of the streaming service. Yi Che, a master candidate in our lab, helped us with the implementation of the DirectShow filters for live video/audio transportation. Jori Liesenborgs from Limburgs Universitair Centrum wrote the JRTPLib[11], an object-oriented RTP library in C++ which is adapted for use in our streaming service.

11. REFERENCES


