

ConferenceXP Research Platform: Toward an Extensible Collaborative Environment

Michel Pahud,
Microsoft Research,
mpahud@microsoft.com

Abstract

ConferenceXP is an initiative of Microsoft Research. The goal of the ConferenceXP research platform (www.conferencexp.net) is to empower researchers to build less infrastructure and concentrate on researching and developing rich collaborative applications for learning and research collaborations. ConferenceXP supports the development of real-time collaboration and videoconferencing applications by delivering high-quality, low-latency audio and video over broadband connections, as well as providing a flexible common framework for designing and implementing collaborative applications. This paper focuses on ConferenceXP as a collaborative platform and the results of early prototypes of advanced real-time applications created using this platform. These prototypes include a collaborative Tablet PC story creation tool for kids, a collaborative Microsoft® Visio® diagram creation tool for technical drawing and a real-time distributed visual assessment tool.

1. Introduction

ConferenceXP enables developers to build a set of interoperable solutions on top of a common framework. With published APIs and a set of base classes, developers and researchers can design powerful new conferencing and collaborative environments, create custom interfaces, and integrate ConferenceXP with existing conferencing and classroom systems. The difficult parts of developing collaborative applications — forming groups, managing group state and status, dealing with network errors — are taken care of by the ConferenceXP platform and its services.

ConferenceXP is a .NET-based platform that employs a peer-to-peer architecture. Because no server is involved, this architecture makes deployment easy, and prevents network traffic bottlenecks and single points of failure. This architecture uses multicast to ensure efficiency and scaling for multipoint video conferencing over a broadband network, as well as for sharing documents and ink over an 802.11b wireless network. It also supports unicast, via point-to-point or a bridge, for environments where multicast is unavailable.

2. Architecture

The ConferenceXP architecture is divided into four logical layers: Network Transport, Conference API, ConferenceXP Capability, and the ConferenceXP application (figure 1).

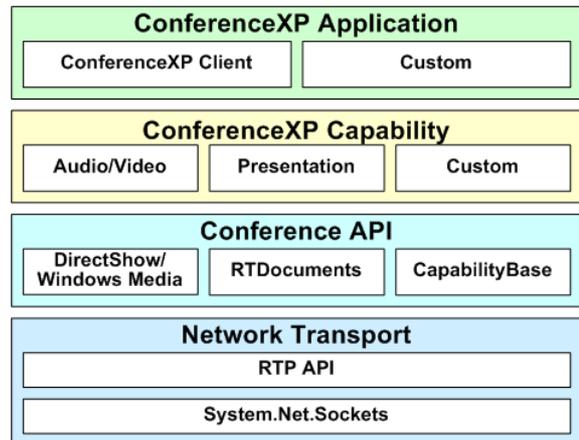


Figure 1: ConferenceXP architecture.

2.1 Network Transport Layer

The Network Transport layer provides a custom implementation of the Real-time Transport Protocol (RTP), based on the managed implementation of Windows Sockets (System.Net.Sockets).

The RTP peer-to-peer network transport is an Internet Engineering Task Force (IETF) standard for audio and video transmission [1]. It is designed for scenarios where low latency is required, such as high-performance conferencing.

To help prevent data loss in difficult network environments, such as wireless networks in large classrooms, ConferenceXP has implemented forward error correction (FEC) algorithms. In addition, we are currently working on implementing a retransmission mechanism.

2.2 Conference API Layer

The Conference API layer enables developers to quickly and easily create add-ons or plug-ins (known as capabilities) and applications without concern for the networking.

CapabilityBase encapsulates the functionality required from other parts of the conferencing layer to create new collaborative capabilities.

The *RTDocument* API provides applications and capabilities with a standard protocol to transfer documents and ink strokes. By using the *RTDocument* protocol [2], applications and capabilities that handle documents and ink can interoperate with each other. The *RTDocument* protocol is a Microsoft .NET runtime implementation of the IMS/SCORM [3] interchange specification.

ConferenceXP's Managed DirectShow API provides a .NET wrapper around DirectShow and Windows Media APIs so that ConferenceXP applications and capabilities can use multimedia devices to send/play data over the network.

2.3 ConferenceXP Application and Capability Layers

The ConferenceXP Application and Capability layers provide the user interface for ConferenceXP. The ConferenceXP Client application enables you to interact and collaborate with others in a virtual collaborative space, called a venue¹. Capabilities are add-in components that add functionality to a ConferenceXP client application. Both ConferenceXP applications and capabilities use the Conference API. The ConferenceXP Capability layer includes the Audio/Video and Presentation capabilities that are included with ConferenceXP.

3. ConferenceXP Capabilities

Now let's take a look some examples of collaborative applications that we have created on the top of ConferenceXP.

As mentioned earlier, the *CapabilityBase* component allows developers to easily create new collaborative capabilities. In fact, creating a collaborative application on the top of ConferenceXP essentially comes down to using simple methods and events (e.g. *SendObject* and *ObjectReceived*). Figure 2 shows the communication flow between capabilities inside ConferenceXP.

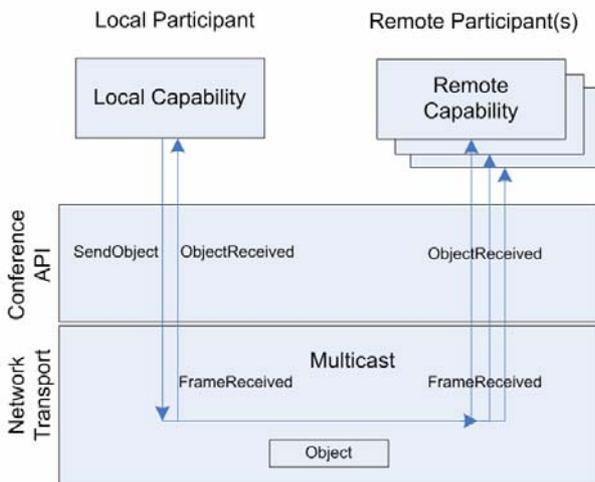


Figure 2: *Capability architecture.*

Using the reflection/metadata feature of .NET, ConferenceXP can dynamically discover capabilities that are co-located with the ConferenceXP client. In addition, when a participant launches a new capability locally, ConferenceXP launches the corresponding capability on remote participants' machines in the same venue

¹ On the network, a venue is a multicast endpoint defined by an IP address and port.

(ConferenceXP assumes that capabilities are installed on each participant's machine and degrades gracefully if this is not the case).

After a capability has been launched, every time a participant's capability calls the *SendObject* method, an object is sent to everybody in the venue, including the sender itself (figure 2).

There are two types of capabilities: *Capability Channel* and *Owned Capability*. When a *Capability Channel* is launched, instances of the capability will continue to run on each participant's machine (in the same venue) until everyone has left the venue. This behavior is useful in scenarios where the initiator starts the collaboration but might leave when the collaboration is still active. For example, you could imagine a chat or a shared whiteboard session where the person who initiated it might leave before the end.

The behavior of an *Owned Capability* is that the person who initiates the capability retains control over when the capability terminates on all participants' machines, either through stopping the capability, or exiting the venue.

The capabilities presented in this paper are *Capability Channels*.

4. How Do I Create a Collaborative Capability for ConferenceXP?

In order to demonstrate that capabilities can be easily created on the top of ConferenceXP, this section will show you the steps for creating a very simple Chat capability. After that we will explain what it takes to create a capability that allows the exchange of ink strokes. Later in this paper we will explain how we created more complex capabilities. This paper is not intended to be a tutorial or a step-by-step guide, but after this section you should have a very good feel for how to create a capability. If you want more details about the capabilities shown in this section, please take a look at the ConferenceXP source code that can be downloaded from www.conferencexp.net.

The first example of a capability that we are going to see is a simple Chat capability that allows you to exchange text messages. The Chat capability will contain a txtSend textbox to enter a message, a btnSend button to send the message and a txtReceive textbox to see the Chat thread. Figure 3 shows the Chat capability user interface.



Figure 3: *Chat capability user interface.*

ConferenceXP is implemented in C#, but you can use any .NET language to create capabilities. Figure 4 shows the Chat capability code.

```

1  [Capability.Name("Chat")]
2  [Capability.PayloadType(PayloadType.xApplication1)]
3  [Capability.FormType(typeof(ChatFMain))]
4  [Capability.Channel(true)]
5  public class ChatCapability
6  {
7      ...
8      void btnSend_Click(object sender, System.EventArgs e )
9      {
10         this.SendObject(txtSend.Text);
11     }
12     ...
13     void objectReceived(object o, ObjectReceivedEventArgs orea)
14     {
15         if (orea.Data is String)
16         {
17             txtReceive.Text = orea.Participant.Name + " : "
18             + (string)orea.Data;
19         }
20     }
21     ...
22 }

```

Figure 4: Chat capability code in C#.

4.1. Creation and Initialization of a Simple Chat Capability

When creating a Windows Form-based capability (which is the case for the Chat capability), you need to first create a Windows application project with Microsoft Visual Studio .NET, and then create the user interface (as shown earlier in figure 3). You then need to add references to the ConferenceXP DLLs (Conference, ConferenceXPInterface, MSR.LST.Net.Rtp, and NetworkingBasics). You also need to add a ChatCapability class in order to initialize the capability.

Lines 1–4 of figure 4 show the code to set attributes to the ChatCapability class. Basically you need to set the following attributes: the name of the capability, the payload type, the name of the Windows form to be launched, and whether the capability is a *Capability Channel* or an *Owned Capability*.

During the initialization phase, you can also specify whether or not you want to allow separate user interfaces for the Sender and Receiver.

4.2. Implementing Send and Receive Object of the Chat Capability

Now we need to implement the string exchange in the Chat capability. ConferenceXP has a *SendObject* method

and an *objectReceived* event handler that allow developers to easily create capabilities that send/receive messages in a multicast group while hiding the complexity of RTP layer, managing sessions/venues/participants and remote launching of capabilities. This is where ConferenceXP really helps you to concentrate on your collaborative capability development, and allows you to quickly create new capability prototypes. Note that capabilities developers can actually choose between a synchronous and asynchronous version of the *SendObject* method.

From an implementation point of view, you simply need to send the text entered in the *txtSend* textbox by using the *SendObject* method when the *btnSend* button is clicked; Lines 7–10 of figure 4 show you the corresponding code. You now need to implement the *objectReceived* event handler in order to get the messages and display them on *txtReceive* textbox; Lines 11–18 of figure 4 shows the corresponding code.

Notice that *orea.Participant.Name* allows you to get the name of the sender. The Chat capability example only uses this information to display the participant name with each message, but you can imagine many uses, including various types of filtering based on a participant's name. In ConferenceXP every participant also receives their own messages because they are part of the multicast group. This is the behavior we want for Chat capability; however you might need to filter out your own messages in more advanced collaborative application; for example if you have a shared whiteboard you definitely want to filter out your own stroke to avoid having the stroke drawn twice on the sender side. If you need to filter out your own strokes, you can do so by comparing *orea.Participant* that contains the owner of the message received with the local participant *Conference.LocalParticipant*.

The member variable *orea.Data* allows you to check the object type received (line 13 in figure 4). In the Chat capability example the objects sent are string objects only, but in a more advanced collaborative capability you should create custom objects. If your capability uses strokes or documents you can use the *RTStroke* and *RTDocument* classes, which enables your application to interoperate with other capabilities that also use these classes. We will get back to this later on in the paper.

This is basically all you need to do to create a Chat capability. Now, let's take a look at some more advanced capabilities that support ink.

4.3. Capabilities with Ink Support

The Tablet PC is a great tool in collaborative environments because of the natural interaction that they offer. You can imagine a lot of ink-oriented collaborative applications.

You need to use 2 additional components to create capabilities with ink support: The Tablet PC API and the ConferenceXP *RTStroke* object.

Tablet PC API

We will first give you a little bit of background on the Tablet PC SDK events and methods that are commonly used for collaborative applications. If you are already familiar with Tablet PC development, please feel free to skip this section.

In your capability application you need to add a reference to *Microsoft.Ink*, add the using statement to it, and initialize an *inkCollector* object.

After that you can use one of the Tablet PC events: *NewPackets*, *Stroke*, and *StrokesDeleting*. All of these events give you the corresponding stroke (or strokes) through event arguments. Here is a summary of Tablet PC events you would need to handle when creating an ink-oriented collaborative application:

- **NewPackets event** - This event occurs many times while a stroke is being drawn. The *NewPackets* event should be used when a collaborative application needs to have the strokes appearing in real-time in the remote participant's machines while there are drawn.
- **Stroke event** - This event is fired only once at the completion of a stroke. The simplified sample presentation application that come out-of-the-box with ConferenceXP uses *Stroke* events.
- **StrokesDeleting event** - This event is fired when the user deletes a stroke (and before the stroke is actually deleted).

In addition, you need some Tablet PC methods to programmatically add or remove strokes on the remote participant's machine:

- **AddStrokesAtRectangle method** - This method can be used to programmatically add a stroke on the remote machine(s).
- **DeleteStroke method** - This method can be used to programmatically delete a stroke on the remote machine(s).

ConferenceXP RTStroke Support

ConferenceXP has an *RTStroke* class to help you manage ink-oriented collaborative capabilities. You can place the stroke into the *RTStroke* object to be able to interchange strokes with other ink-oriented capabilities. In addition, using an *RTStroke* object allows you to assign a GUID to each stroke, which is needed when you want to implement stroke deletion in a collaborative environment. The *RTStroke* class also takes care of stroke serialization for you. There is a special object named *RTStrokeRemove* that allows you to request stroke deletion.

Using the Tablet PC API and ConferenceXP *RTStroke* class allows you to easily create ink-based collaborative capabilities; you simply have to add the code to call the *SendObject(RTStroke)* method to send the stroke object when the *Stroke* (or *NewPackets*) event is fired. Similarly when a stroke is deleted, the Tablet PC event

StrokesDeleting is fired, so you can call the *SendObject(RTStrokeRemove)* method to inform the remote participants that the stroke deleted no longer exists.

Figure 5 shows how a whiteboard capability works: steps 1–4 show what happens when participant A draws a first stroke (with GUID1); steps 5–8 show what happens when participant A draws a second stroke (with GUID2); steps 9–12 show what happens when participant B deletes the first stroke of participant A (the one with GUID1).

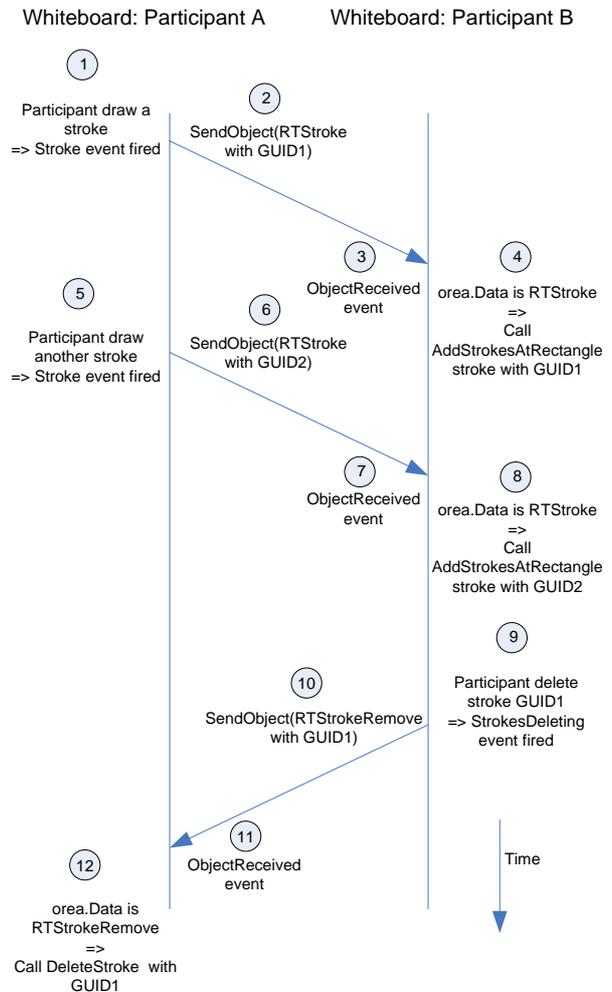


Figure 5: Stroke exchanges on the top of ConferenceXP.

ConferenceXP has also a class named *RTDocument* that enables interoperability between document-based capabilities. This class describes documents with metadata, a table of contents and associated pages. Like strokes, each page has a unique GUID. The presentation capability (also known as “CXP Presentation”) that is installed with the ConferenceXP client is a good example of a capability that uses *RTStroke* and *RTDocument* class.

4.4. Audio/Video Capabilities

The collaborative capabilities are not CPU or bandwidth intensive; however Audio/Video capabilities are CPU and bandwidth intensive. For performance reasons the Audio/Video capabilities do not use *SendObject* and

objectReceived, but instead use custom DirectShow filters to access the network directly.

ConferenceXP is capable of 5-way videoconferencing in full screen and 30fps with a dual processor 3.0Ghz workstation. Figure 6 shows a snapshot of a 4-way videoconference running.



Figure 6: Snapshot of a 4-way videoconference.

5. Prototypes of Advanced Collaborative Capabilities

So far we have shown you examples of simple capabilities in order to explain the basics of ConferenceXP's capability support. To validate ConferenceXP as a platform that enables the creation of rich collaborative applications, we created more advanced applications in various fields: a collaborative Tablet PC story creation tool for kids, a collaborative Microsoft® Visio® diagram creation, and a distributed visual assessment tool.

5.1. A Collaborative Tablet PC Story Creation Tool for Kids

This capability, adapted from a stand alone application, allows kids to collaboratively create scenes of a story by placing, sizing, and rotating 3D objects, adding dialog boxes, and painting it (figure 7). In order to make this application collaborative on the top of ConferenceXP, we just needed to modify the strokes behavior of the stand alone version in order to exchange them as described earlier in section 4.3. In this capability we suppose that each participant has the 3D graphics objects library, so we created an additional custom object to embed the position/rotation/size of 3D objects in order to be able to communicate this information in real-time through ConferenceXP using the *SendObject* method. We also use color-coding to determine if the 3D object displayed on the scene has been created by the local or remote participant; we can know this information using *Conference.LocalParticipant*. And that's it! The modifications to turn this stand alone application into a collaborative capability were done in only a few hours.

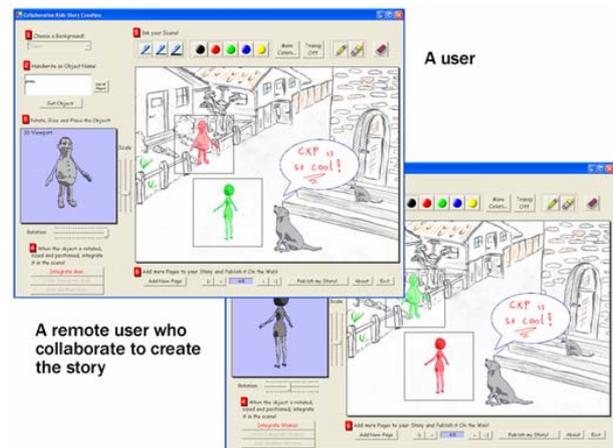


Figure 7: A collaborative Tablet PC Story Creation Tool.

5.2. A Collaborative Visio Capability

Similarly, we created a Visio capability where the Visio shapes are shared among participants. Besides writing a custom mapping for describing Visio objects, we also implemented per-participant color-coding, as well as remote participant shape hiding (figure 8).

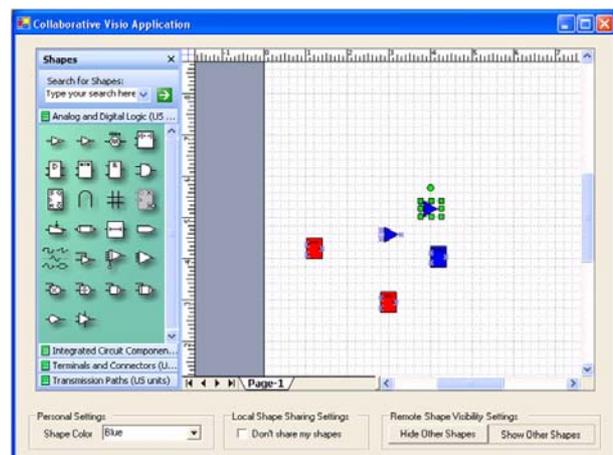


Figure 8: Snapshot of the collaborative Visio capability.

5.3. A Distributed Visual Assessment Capability

We created a distributed visual assessment tool based on XML formatted assessment games description that a professor can generate with several authoring tools; so far we have a Natural Language Processor (NLP) authoring tool to create language learning games and a wizard to generate other types of assessment games. We designed a custom XML object representing the game as well as a custom object representing the score. Figure 9 shows the process where a professor creates and distributes an assessment game using our custom XML object (steps 1–6) and where students play and the score is returned to the professor in real-time using our custom score object (steps 7–9).

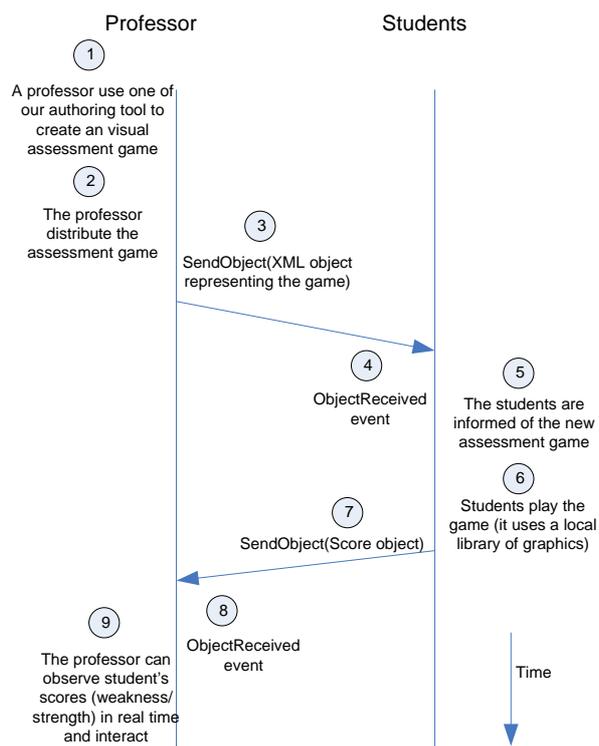


Figure 9: A distributed assessment tool.

6. Improvements and Challenges

A next step would be to use generic constraint solvers in order to allow collaborative participants to add constraints on their environment. Constraints enable you to minimize the communication traffic between participants. A trivial example of use for generic constraint solvers would be to inform a participant working on designing the outside part of a car in a Visio technical drawing that another participant is designing the engine of the car bigger than the outside of the car. In this simple scenario you can easily imagine that constraints would decrease the communication between participants (because the designer of the outside part of the car doesn't have to periodically ask the engine designer the size of the car engine).

7. Related Work

Access Grid [4] is another interesting collaborative platform that has a lot in common with ConferenceXP. The audio/video aspects of ConferenceXP and Access Grid are very similar – they both use a peer-to-peer architecture, multicast over RTP, and support the concept of virtual venues.

Like ConferenceXP, Access Grid also provides support for non-A/V collaboration, through support for Shared Applications, which are analogous to Capabilities in ConferenceXP. While the goals and applications are quite similar, the architectural design for ConferenceXP Capabilities and Access Grid Shared Applications is quite different. ConferenceXP uses the same network transport (multicast over RTP) to support both A/V and Capabilities.

In contrast, Access Grid uses a client/server, TCP/IP architecture for Shared Applications.

Another difference between ConferenceXP and Access Grid is that Access Grid is designed to persist more state and application data (at the Venue Server), while ConferenceXP is designed with ease of deployment in mind, and does not require a complicated server infrastructure.

ConferenceXP has been successfully used in many collaborative projects between Microsoft Research and universities. Many innovative capabilities have been created by universities, including Classroom Presenter at the University of Washington [5] and ReMarkable Texts at Brown University [6]. More information about collaborative work with universities can be found in the Research section [7] on the ConferenceXP community site.

8. Conclusions

You can imagine thousands of innovative collaborative applications that would fit in real-world scenarios. However, you can not fully validate your ideas without creating working collaborative prototypes and having participants using it. We showed you in this paper that the ConferenceXP research platform is a powerful tool to enable researchers to very quickly and easily create new innovative advanced real-time collaborative applications in order to validate ideas with participants in real-world scenarios.

Acknowledgements

We would like to express gratitude to all strong contributors on the Microsoft Research team that make ConferenceXP a great success: Chris Moffatt, Jason Van Eaton, Jay Beavers, Patrick Bristow, and Tim Chou. Without them, ConferenceXP would not be here today. In addition, we are grateful to the University Relations group at Microsoft Research for supporting this project.

We would like also to express our gratitude to Takako Aikawa and Lee Schwartz from the NLP Research group for their great contribution in the NLP assessment game generator.

References

- [1] IEFTE specification for RTP: <http://rfc.net/rfc3550.html>
- [2] ConferenceXP RTDocuments Specification <http://www.conferencexp.net/community/library/RTDocsSpecification.htm>
RTDocuments Talk, given by Jay Beavers
http://www.conferencexp.net/community/library/producer/RTDocumentsTalk/RTDocumentsTalk_files/Default.htm
- [3] IMS standards <http://www.imsproject.org>.
Advance Distributed Learning (ADL) initiative and Sharable Content Object Reference Model (SCORM) www.adlnet.org.
- [4] Access Grid site <http://www.accessgrid.org>
- [5] <http://www.cs.washington.edu/education/dl/presenter/papers.html>
- [6] <http://www.cs.brown.edu/research/graphics/research/ReMarkableTexts>
- [7] Research section of ConferenceXP community site <http://www.conferencexp.net/community/Default.aspx?tabindex=4&tabid=67>