MULTIMEDIA STREAMING PLATFORM BANDWITH CONTROLL CONGESTION DETECTION AND BANDWIDTH ADAPTATION

Ovidiu Răţoi[1]
Haller Piroska[2]

Abstract

We propose a platform for distributed multimedia systems. The proposed platform is implemented using the Netscape Portable Runtime (NSPR) and the Cross-Platform Component Object Model (XPCOM). This ensures system portability, flexibility and performance. The platform is equipped with a congestion detection algorithm and a bandwidth control mechanism thus controlling the transfer rates between the communication parties. Using this kind of bandwidth management this platform allows real-time streaming across multiple networks.

Keywords — *real-time multimedia platform; XPCOM components; congestion detection; bandwidth control;*

Introduction

Recent years have shown an increased interest towards multimedia rich applications. Multimedia content ranges from text or simple images to audio and video data or even animations. The increased availability of broadband Internet connections leads the way towards applications that offer high quality multimedia streaming over wide area networks. In this context there is a need for solutions that enable application developers to quickly and effortlessly develop this kind of applications.

In the same time software components technologies and component based software engineering are maturing, in fact the use of components is a sign of maturity in any field of engineering. The usage of software components offers a lot of advantages the most important of them being reusability, a component once developed may be reused in any number of applications, depending on how generic are the services it offers. Also the task of applications developers changes from development of new software to composition of existing pieces. Another characteristic property of software components is encapsulation. This property hides the internal structure and exposes a well defined interface through which the services are accessed. Encapsulation confers high flexibility to component based software as individual components can be easily replaced with improved ones as long as their interface remains identical.

Network communication was always an important issue when handling multimedia content especially when real time streaming was involved. A research team tackled this problem and proposed a multimedia applications middleware which regulated network

[1] Ratoi Ovidiu, drd. prep. ing., "Petru Maior" University of Tg. Mureş, email: oratoi@engineering.upm.ro
[2] Haller Piroska, Conf. dr. ing., "Petru Maior" University of Tg. Mureş,email: phaller@upm.ro

traffic, optimized resource usage and offered a high degree of portability to applications [1].

Recently some groups are proposing a platform for collaboration systems which integrates mobile devices [2]. It uses the client-server architecture to provide multimedia content adapted to the capabilities of any devices used clients.

Our goal was to create an interactive Web application based on components that allows bi-directional, real time communication between the resources and the user. This paper describes the component based platform proposed and implemented by us for multimedia transfer.

The paper is structured as follows. In section 2 we provide a short description of the Mozilla platform. In section 3 we provide a detailed description of the proposed platform. One adaptive control mechanism is presented in section 4 along with some test results. We end the paper with a conclusion and future work in section 5.

Mozilla Platform architecture

Mozilla is an open source portable platform, developed and maintained by the Mozilla Foundation, best suited for rapid development of highly interactive visual applications [3]. Its conceptual architecture is presented in *Figure 1*.

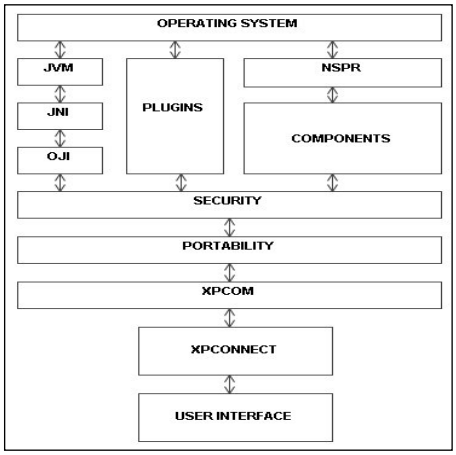


Figure 1. Mozilla Platform Architecture

Mozilla based applications have three possibilities to access the Operating System: using the JVM (Java Virtual Machine), using plugging or through an API called NSPR (Netscape Portable Runtime). NSPR is a portable API designed to provide operating system level services like threads and synchronization support, file and network I/O, memory management, time management, atomic operations or process creation.

XPCOM (Cross Platform Component Object Model) is Mozilla's object management and discovery system, very similar to Microsoft COM and remotely to CORBA (Common

Object Request Broker Architecture). It was designed to provide greater flexibility to the platform and to applications developed on top of it. These components can be created in a variety of languages ranging from C, C++ to JavaScript or Python and are accessed through a set of interfaces they implement [4]. In order to provide greater portability and implementation language independence, interfaces are described in a special language called XPIDL (Cross Platform Interface Definition Language), a variant of CORBA IDL. Object lifetime management and interface discovery are implemented in a Microsoft COM style, by reference counting and special methods for querying all implemented interfaces.

XPConnect is the technology used to expose object interfaces to scripting languages, like JavaScript. User interfaces for applications are usually described in XUL (Mozilla's XML based User interface Language) [5] or HTML the well-known markup language.

Using the Mozilla platform, we focused on the development of streaming components, capable of receiving multimedia data from different sources, decode it, and deliver it to the user interface. The modular architecture of the Mozilla platform enables developers to add or remove modules with little effort, fitting the software to the available hardware and adjusting functionality to match product requirements. Our components adjust the transfer rate continuously, monitoring the devices and network capabilities. The need of the self-managing components was recently introduced in Web technologies [6], but not in implementations.

Platform for Distributed Multimedia Applications

Platform model description

A well known method for providing a high degree of transparency and portability to distributed applications is positioning an intermediate layer called by us *Multimedia Platform* between the operating system and the application. In *Figure 2* the multilayer structure of a multimedia applications platform is presented.

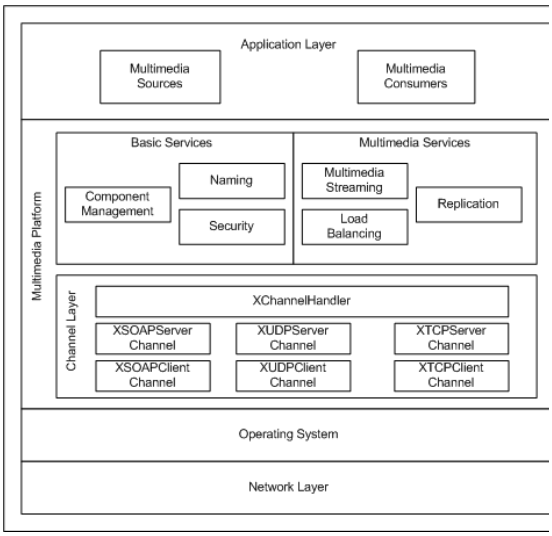


Figure 2. Multimedia Platform Architecture

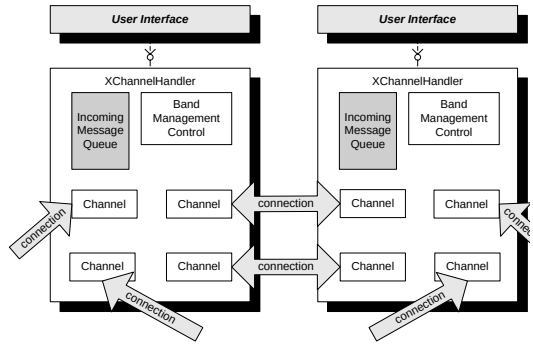


Figure 3. Platform Connection Model

The bottom layer in this architecture is represented by the network which provides host computer interconnection and basic data transmission services. On the following level we have the operating system which provides services ranging from process management or memory organization to communication and synchronization. Above the operating system we can find the *Multimedia Platform* which is divided into two sections: a channel layer (described later in this paper) and a service layer.

On top of the *Multimedia Platform* there is the application layer which contains multimedia sources or multimedia consumers. The Multimedia Platform offers a service for data streaming between multimedia sources and consumers. Whenever a consumer needs data from a source a stream between them has to be established. The communication is based on the concept of channels.

A channel, as mentioned in the previous section, is a logical communication link between two software entities, like presented in *Figure 3*. The communication requires the existence of a connection between each pair of communicating applications. Channels embody communication protocols, while access is provided through one single interface.

The *Multimedia Platform* provides an interface for using the channels. By using this interface we can create or destroy a channel, send messages to one specific channel and get the received messages from opened channels. Each newly created channel is given a unique channel ID. All operations involving channels are done through this ID. This interface exposes four functions used for creating and destroying channels, sending messages to one channel or receiving messages from the opened channels.

The platform was designed for usage in Mozilla web based client application also. In this case some elements, like receiving channel events from the platform, had to be taken into consideration. Due to restrictions imposed by the *XPConnect* technology notifications

cannot be sent asynchronously from the component to the user interface thus an alternative solution had to be found. One feasible option would be to implement a waiting queue in the component for stream data or events, and then at regular time intervals the user interface would query the object. To prevent memory allocation overflow this queue had to be limited to a maximum size.

A more detailed description of the channel architecture provided by the Multimedia Platform it is provided in a previous article [7]

Channel traffic load

Once the platform was operational we tested it for maximum traffic load. For this purpose, one client and one server application was designed based on the proposed platform. Both applications ware developed as standalone applications and neither of them had a graphical interface. The purpose of those two applications was to exchange messages at maximum speed. Once the messages ware received, they were extracted from the platform and erased as quickly as possible, with no other processing made.

The tests were made over the internet using two Windows machines on a period of 160 minutes with a 1 minute sampling time. For testing purposes we used the TCP type of channels. The results are shown in *Figure 4* and *Figure 5*.

The differences between the two tests are in the size of the packages send through the channels. The first test used 1024 B packets and the second one used packets 10 times bigger. The spikes on the graph appeared when the internal queue size reached the maximum value and, as a failsafe measure, the platform stopped reading data from the channels. In this case, as we can see on the graph also, the channel traffic load showed a decrease until some of the incoming messages were processed. After the internal queue size had dropped under that critical value the channel data reading started again and the measured band has increased again. As we can see from the graphics the maximum traffic load over the internet using the current architecture of the platform is in the range of 4000 to 5000 Kb of data. This value is a satisfying one for a client-site application used in real time multimedia streaming, and especially for a web-based client application.

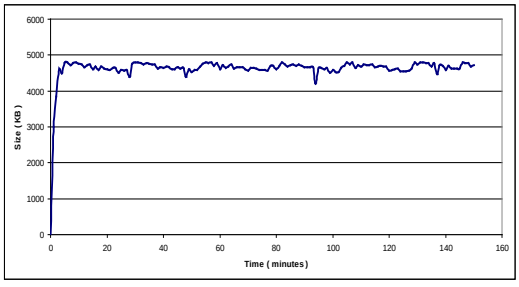


Figure 4. Test results for 1024 B packets with 1 minute time interval

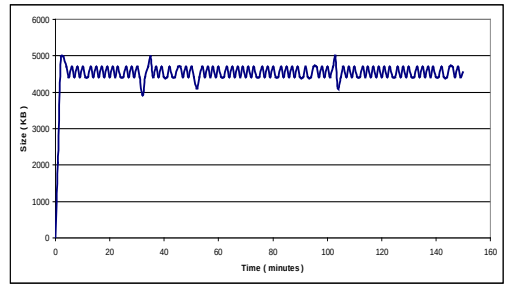


Figure 5. Test results for 10024 B packets with 1 minute time interval

Video streaming application test results

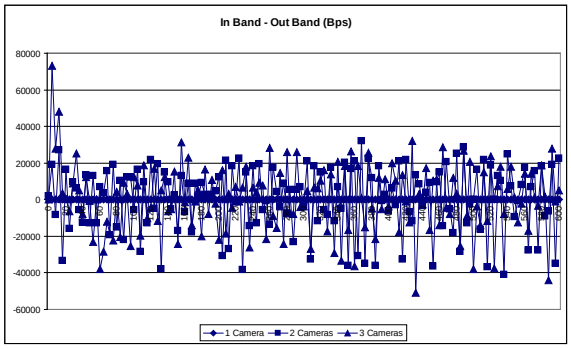
For testing the platform in a multimedia environment we had created a stream server running in *XULRunner* and a web-based client application running in "Mozilla Firefox 2.0.0.20". Both of them used the proposed multimedia platform. For the client application, the interaction between the browser and the platform was made using Java Script. For testing purposes we opened several instances of the client application that were connected to the stream server. Several sources were also connected to the stream server and the client applications received frames from them.

Using the model presented above, the video streaming application was tested on several platforms with variable number of cameras. Three parameters were measured, ***Incoming Bandwidth*** resulting from data received on the communication channel established with the stream server, ***Outgoing Bandwidth*** resulting from the total size of the video frames transmitted to and displayed by the user interface and ***Queue Size*** representing the number of video frames stored in the object's waiting queue. All tests were conducted with the same application, stream server and cameras on a period of 10 minutes with a 4 seconds sampling interval. Once again the TCP based channels were used again. The results are presented in the following figures.

Test results show that the application performs well on both **Windows** and **Mac OS** environments and although there are oscillations in bandwidth, the waiting queue never grows bigger than two frames which, considering a data rate of 7-10 fps from each camera, translates into a very small delay, even when receiving stream from three different cameras. The performance of the same application is significantly worst in the **Linux** environment especially when video stream is received from more than one camera.

The operating systems are not responsible for the different performances of the platform. As a matter of fact, the ***Incoming Bandwidth*** on all of the tested platforms was in the same range. The differences were because of the ***Outgoing Bandwidth***. Mozilla Firefox has a different behavior on those environments when rendering frames.

Figure 8 shows that when displaying images from three different cameras the difference between incoming and outgoing bandwidth is quite high, which produces an abrupt accumulation of frames in the waiting queue, as it can be seen from *Figure 9*. From this increase of the waiting queue size results an unacceptable delay in the video stream.



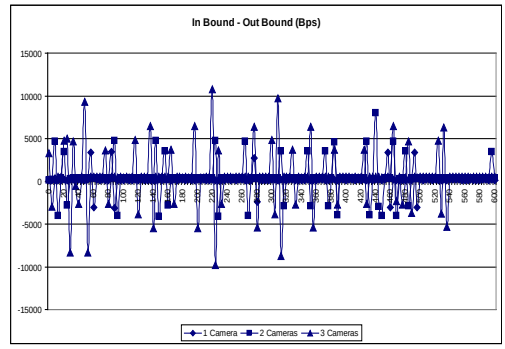
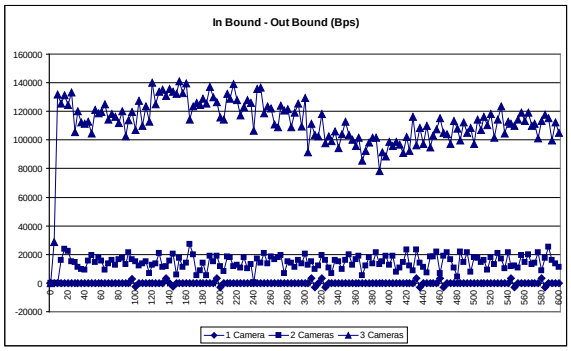


Figure 6. Controlled bandwidth on Linux

Figure 7. Bandwidth on Mac OS X



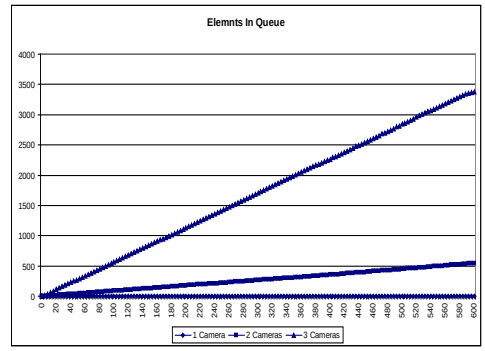


Figure 8. Bandwidth on Linux

Figure 9. Queue size on Linux

Adaptive controll mechanism

For using an adaptive stream control, the stream servers need to have a mechanism for setting the prescribed value for client bandwidth. Some of them accomplish this by recompensing the multimedia stream accordingly to the prescribed value. Other servers accomplish this by dropping some of the frames that should be sent to the client.

Exploiting this facility could improve application's performance on some platforms by reducing delays in stream, especially when a large number of devices are observed. Because the internet bandwidth can vary in time and the application is an interactive one where the number of devices from which stream is received could also vary in time, an adaptive control mechanism has to be implemented. A possible solution would be to introduce a new *XPCOM* component responsible for gathering parameter values measured by the channels and taking control decisions according to them. In the proposed platform model this component has a passive role. Every active channel will report periodically to it some parameters.

One version of the adaptive control mechanism is presented in a previous paper of the authors [8]. In the previous presented algorithm the channels reported periodically 3 parameters: ***Incoming Bandwidth***, ***Outgoing Bandwidth*** and ***Queue Size***. This approach could not prevent the line congestion, but once the congestion was there the adaptive algorithm will decrease the prescribed value thus trying to end the congestion.

A new approach was developed that implements one congestion detection algorithm.

Congestion detection algorithm

The algorithm developed for the detection of the line congestion is using the *Ping-Pong* strategy. One exchange of messages is taking place between the two endpoints engaged in a communication. The messages exchanged between them are short messages containing only the timestamp of the messages as described *Figure 10*

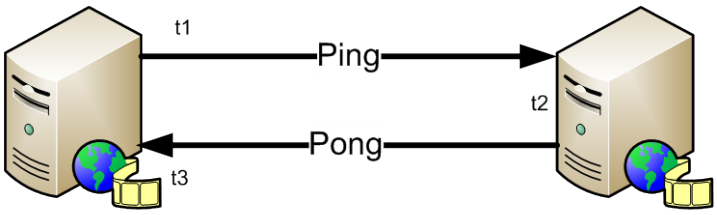


Figure 10. Ping-Pong message exchange

As shown in *Figure 10* the PING message has t_1 timestamp. This timestamp value will be appended to the PONG message which in turn has t_2 timestamp value. The moment on which the PONG message is received is given the t_3 timestamp.

Based on this type of message exchange, the *Ping-Pong* congestion detection algorithm has two stages.

- The first stage computes the average RTT (Round Trip Time) and the ΔT value, which represents the time difference between the two computers. This stage is finished when a certain number of epochs are accomplished.
- The second stage of the algorithm uses the RTT and the ΔT values for detecting the collision on the communication link

The first stage

The average round trip time is the time difference between the moment the PING message was send (t_1) and the moment the PONG message was received (t_3).

The current value of RTT (Round Trip Time) can easily be calculated as t_3-t_1 for every acknowledge packet received, where t_1 represents the moment the PING message was send and t_3 represents the moment the PONG message was received. To determine its average value, an exponentially weighted moving average (EWMA) filter was applied:

$$RTT_k = \alpha\, RTT_{k-1} + (1-\alpha)(t_3 - t_1)$$

One small value of α corresponds to an agile filter while large values of α corresponds to a more stabile filter. The most recent TCP-friendly rate control protocol specification (Handley, Floyd, Padhye, & Widmer, 2003) recommends a default value of 0.9.
The current value of ΔT can easily be calculated as the absolute value of $(t_1+t_3)/2 - t_2$.

$$\delta T_i = \; abs\left(\frac{t_1 + t_3}{2} - t_2\right)$$

To determine the ΔT value representing one accurate time difference between the two computers the minimum value should be taken into consideration:

$$\Delta T = \min_{i=1..n} (\delta T_i)$$

The seccond stage

The second stage is using the same Ping-Pong exchange of messages. Once the RTT and the ΔT values are computed we can compute the ST and RT values, where:

- ST – represents the time needed by one message to be send

$$ST = \text{abs}(t_2 - t_1) - \Delta T$$

- RT – represents the time needed by one message to be received.

$$RT = abs(t_3 - t_2) - \Delta T$$

Based on those values we can predict if the communication link is starting to be congested or if the bandwidth could be increased. The congestion control algorithm is formulated as follows:

```
if ( ( ST < 0 ) || ( RT < 0 ) )

{

        // the computed ΔT is wrong

        // recalculate ΔT value by going to STEP 1

}

if ( ST >> 0 )

{

        // there is congestion detected on the send line

}
```

If the ST or the RT values are negative ones, than the ΔT and the RTT should be recalculated because the time difference between the two endpoints has changed or the last computed value of ΔT was not exact.

Bandwidth control mechanism.

The bandwidth control algorithm is working in correlation with the *Congestion detection algorithm*. Based on some parameters provided by the channels the prescribed bandwidth values are computed. Those parameters are: *Incoming Bandwidth*, *Outgoing Bandwidth*, *Queue Size*, the *ST* value and the *RT* value.

If the current ST value is much greater than 0 (zero) than congestion was detected on the send line. If this happens than the prescribed value for the outgoing bandwidth must be decreased.

If the current RT value is much greater than 0 (zero) than congestion was detected on the receive line. If this happens than the ***Maximum Incoming Bandwidth*** is decreased. This

value is send to the other entity engaged in the channel communication link. This entity will set the prescribed value for ***Outgoing Bandwidth*** accordingly to the received value, but less than the older value.

Because some multimedia servers accomplish this by dropping some frames, video quality will decrease but there will not be any delays, thus maintaining the real-time quality of the stream.

If the ST value is near 0 (zero) then the ***Outgoing Bandwidth*** could be increased if needed. If the RT value is near 0 (zero) then the ***Incoming Bandwidth*** could be increased. In case this action could be made possible, meaning the ***Queue Size*** permits it, the increased value of the ***Maximum Incoming Bandwidth*** is send to the other entity engaged in the channel communication link.

Some test results are presented in Figure 11 and Figure 12.

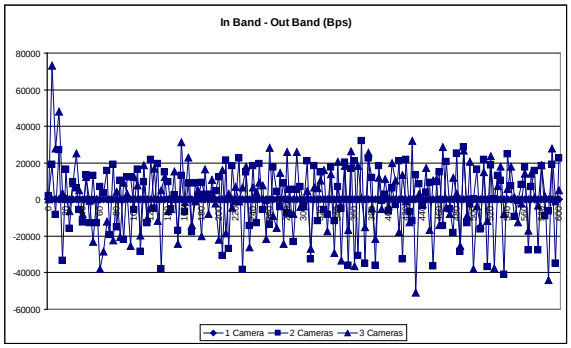


Figure 11. Controlled bandwidth on Linux

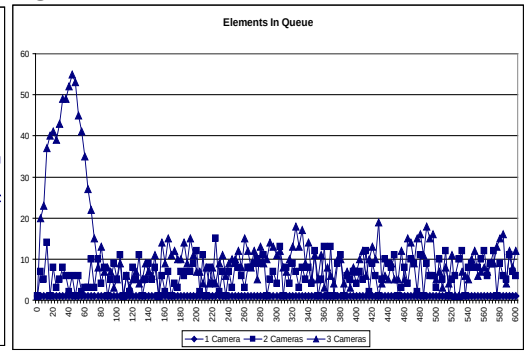


Figure 12. Queue size on Linux with bandwidth control present

Conclusion and future work.

In this paper we proposed a component based, real time streaming, portable, Web platform for distributed multimedia applications, developed on the Mozilla Platform. Based on this model we implemented a set of components that can be used in any kind of multimedia streaming application.

Furthermore a monitoring and control mechanism was presented, which allows the application to dynamically change transfer rates in order to reduce delays in the stream caused by slow presentation rates. One congestion detection algorithm is presented and the bandwidth control mechanism that works in correlation with the congestion detection algorithm.

As future work we intend to extend the adaptive stream control mechanism and to prepare the multimedia platform for usage in a dynamic QoS environment. This means that applications could fine tune the adaptive control mechanism in such a way that different types of multimedia content should be treated different. What this means is the fact that

audio streams could be preferred over video ones or even the other way around if necessary.

References

[1] M. Lohse, M. Repplinger, P. Slusallek, *"An Open Middleware Architecture for Network-Integrated Multimedia"*, Proceedings of the Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems: Protocols and Systems for Interactive Distributed Multimedia, Portugal, pp. 327-338, 2002.

[2] X. Su, B. S. Prabhu, C. C. Chu, R. Gadh, *"Middleware for Multimedia Mobile Collaborative System"*, Proceedings of IEEE ComSoc Third Annual Wireless Telecommunications Symposium (WTS 2004), USA, pp. 112-119, 2004.

[3] Alan Grosskurth, Ali Echihabi, *"Concrete Architecture of Mozilla"*.

[4] Doug Turner, Ian Oeschger, *"Creating XPCOM Components"*, Brownhen Publishing, 2003.

[5] Nigel McFarlane, *"Rapid Application Development with Mozilla"*, Prentice Hall, 2003.

[6] H. Liu and M. Parashar, *„Rule-based Monitoring and Steering of Distributed Scientific Applications"*, International Journal of High Performance Computing and Networking (IJHPCN), issue 1, 2005.

[7] Ovidiu Ratoi, Haller Piroska, Ioan Salomie, Genge Bela, *"Component based platform for multimedia applications"*, Proceedings of the 7th RoEduNet International Conference, pp.40-43, 2008

[8] Ovidiu Ratoi, Haller Piroska, Genge Bela, *"Multimedia streaming platform based on components"*, The 4 edition of the Interdisciplinarity in Engineering International Conference, pp 289-296, 2009.