Journal of Mobile Multimedia, Vol. 4, No. 1 (2008) 042–058

MOBILE MULTIMEDIA FOR MULTIUSER ENVIRONMENTS

DANIEL C. DOOLAN SABIN TABIRCA
Department of Computer Science, University College Cork
Cork, Ireland
{*d.doolan, s.tabirca*}*@cs.ucc.ie*

LAURENCE T. YANG
Department of Computer Science, St. Francis Xavier University
Antigonish, NS B2G 2W5, Canada
lyang@stfx.ca

Received July 1, 2007
Revised October 15, 2007

Mobility especially the flexibility given to us by the mobile phone is the future of computing as we know it. No longer are we restricted to sitting at a desk in front of a powerful desktop machine. Mobile technology of today allows users to work, learn and play no matter where they may be. Wireless technology is becoming more and more a standard feature of computing, so much so that it is expected that approximately two billion Bluetooth enabled devices will have been produced by the end of 2007. This paper examines how Bluetooth application development may be simplified for the programmer by use of the Mobile Message Passing Interface (MMPI). It explores a selection of application areas that can benefit from this simplified means of wireless inter-device communication, including: compute intensive tasks, Mobile Learning and Multi-player gaming.

Keywords: Mobile Message Passing, Bluetooth, Mobile Learning, Multiplayer Gaming

1 Introduction

The demand for mobile technology is phenomenal. In 2005 world wide mobile phone sales outstripped personal computers by a ratio of approximately 4:1, with in excess of 800 million phones shipped [1]. The current demand in the world today puts the annual sales of mobile phones at approximately one billion devices per year. The majority of medium to high end phones come Bluetooth enabled as standard. People are no longer happy with owning just one mobile phone, this is clearly represented with Europe having an average market penetration level of 111% for the year ending 2006 [2]. Luxembourg has been the European leader in the demand for mobile technology for some years now, featuring a record high of 156% penetration in 2005, surprisingly this figure dropped to 138% in 2006. These averages are greater than the European average for personal computers, standing at just 59%.

Just a few short years ago the mobile phone was a device designed solely for the purposes of voice communication, and has evolved to include additional dimensions such as text messaging, multimedia messaging, and integrated cameras. Mobile phones are no longer just simple communication tools but complex devices that have significant computational poten-

tial. More and more do we see mobile phones used for entertainment (gaming), while the introduction of digital television has opened up new avenues of usage.

Such developments are reliant on the continual improvement of the processing and power usage attributes of the mobiles of today. Many off the shelve phones run with processor speeds of 100 to 220Mhz (Nokia 6630, 6680) [3] [4]. October 2005 saw the announcement of the Cortex-A8 [5][6] running at 1Ghz, such is the potential tomorrow's mobile devices.

The release of the Apple iPhone (June 2007) [7] has heralded a new era in mobile computing. A key feature is the removal of the traditional button based keyboard in favor for a touch sensitive onscreen keyboard. It also sports the Mac OS/X operating system. New ways of interacting with the devices are becoming ever more popular, thanks to the addition of additional sensor technology. Phones such as the Nokia 5500 Sport include an accelerometer allowing fitness buffs to monitor their progress. Similar devices also feature in the iPhone to allow the screen to display in portrait or landscape mode based on the devices orientation.

Much of present day research on mobile interaction is focused on the new means of data input by the addition of additional sensors [8] [9] [10] from accelerometers and gyroscopes to compass and temperature sensors. The integration of GPS chips allow for far more accurate tracking of user positions over cell network triangulation. The mobile of tomorrow could very well be more akin to the "Tricorders" of Star Trek that appeared on screen as science fiction not too long ago. Very soon one will be unable to tell the difference between phone, PDA and personal computer, as they will all have similar capabilities in terms of the array of applications that may be executed on them.

On several Star Trek episodes when the main computers weren't functioning it has been said that "we could always network a few tricorders". The Bluetooth enabled phones of today enable us to "network" our devices together providing a platform capable of carrying out complex processing that a single mobile device could not do by itself. This could be because of insufficient resources (memory, battery power), or it could be a problem that would simply take too long from a users perspective to compute.

1.1 Bluetooth

Named after Harald Blatand "Bluetooth" king of Denmark in the mid to late 10^{th} century. Bluetooth [11] [12] provides an effective low power consumption, short range radio frequency based wireless communication mechanism operating at circa 2.4Ghz. The Bluetooth 1.2 specification defines the real throughput at 723Kbits/s. The inclusion of a Java Virtual Machine with just about every mobile phone currently shipped allows for the effective Bluetooth applications to be developed using the JSR-82 Bluetooth package, providing seamless interaction with the underlying Bluetooth communications stack. The Enhanced Data Rate (EDR) ratified in November 2004 provides a maximum throughput of 2.1Mbits/s, and is included as standard with devices such as the Intel based iMacs. With millions of Bluetooth devices shipping per week it is clear that this technology will be around for some time to come, especially with the development of Ultra Wide Band (UWB) Bluetooth [13] that allows for USB2.0 data transmission speeds of 480Mbits/s. Speeds such as this will allow for the real time streaming of High Definition television, and the backward compatibility with the previous standards ensures we can use our mobile devices to interact with everything from our televisions, and set top boxes, to our cars and computers.

1.2 Collaborative Computing

For the powerful desktop machines of today there are many problems that are simply too large and complex to compute. A single system may have insufficient resources such as memory, or the resultant computation could take so long that it is completely unfeasible. Examples where the combined power are essential include problems such as weather prediction, DNA mapping and aircraft aerodynamics simulations. Such problems require the use of dedicated hardware, IBM's BlueGene Project containing 131,072 processors [14] is currently the most powerful system in operation. Not all research centres have the resources to built such massive systems. An alternative is to operate a client / server type model were client applications across the Internet carry the processing using the free CPU cycles of the machines they are executing on. The Search for Extraterrestrial Intelligence (SETI) project [15] is just one example that is part of the Berkeley Open Infrastructure for Network Computing (BOINC) [16]. Mobile devices have far more limited resources that powerful servers and desktop machines, but they still face the same problem, sometimes a problem would just take far too long to compute in a reasonable amount of time on a single device. Therefore with the plethora of mobile devices that are available, the solution lies in using the combined computational power of several devices to solve a problem that would be unfeasible on a single device.

2 The Mobile Message Passing Interface

The Message Passing Interface [17] has been a standard that has benefited parallel computing for 15 years, ever since its introduction in 1992. Many implementations based on the standard exist, most notably MPICH [18], which is a freely available library that provides C and Fortran message passing functions. A Java based implementation called mpiJava [19] provides an object orientated interface to the C based MPICH library through Java Native Interface (JNI) calls. An all Java implementation is Message Passing in Java project (MPJ) [20]. Systems that run any of these implementations are typically very high end machines running on high speed cabled networks.

In the world of mobile computing one does not have the luxury of connecting devices together using high speed cabling such as fiber. Therefore the MMPI implementation is a combination of the MPI paradigm and Bluetooth technology to provide the underlying inter device communications mechanism. The most common form of Bluetooth network is Point to Point, and is readily used for hands free communication, needing only the pairing of the hands free kit with the mobile phone. Other such point to point examples include wireless keyboards, mice and file transfer between devices. The MMPI [21] [22] library is based on the Piconet topology, where by a maximum of eight devices may be interconnected. This interconnection takes the form of a Star network where one central node (the Master) can see all the Client nodes, but Clients are unable to directly communicate with each other. The Star topology is unsuitable for message passing as every node must be capable of communicating directly with any other node in the system. Therefore for the Piconet strategy to be an effective platform for the MMPI it is essential to transform the Star network into a fully interconnected mesh topology.

One other Bluetooth topology exists, that of the Scatternet. This structure allows for the interconnection of two or more Piconets together by means of a bridging device between the two smaller networks. It allows for the creation of networks far larger that the maximum

limited size of eight for the Piconet, but cross Piconet communication is bottlenecked by the bridging node. A clear example of this is when a device in one Piconet wishes to broadcast to all devices. The bridging node will have to relay on the message to all devices in its corresponding Piconet.

The MMPI implementation is centred around three base classes (Figure 1). All of the message passing functions are handled by the MMPI class, this includes methods such as send(...), recv(...) and bcast(...). The two remaining classes are helper classes that facilitate the creation of the processes of device / service discovery and the creation of the mesh interconnect. All communication within a Bluetooth network uses the Client / Server architecture and is reflected in the classes BTClient and BTServer. On execution of any MMPI based program the user must select a mode for the device be it Master or Slave. This requirement is necessary so that on instantiation of the MMPI object the required underlying BTClient or BTServer may be created. In addition to the communications classes the MMPE class allows for parallel graphics to be undertaken, similar to the graphics functions of the MPE extension to the MPI standard. Unlike MPI based graphics where all of the drawing is carried out on the screen of the root node, the MMPE system is capable of both root node drawing and also broadcast drawing, where the screens of all devices may be updated in unison.

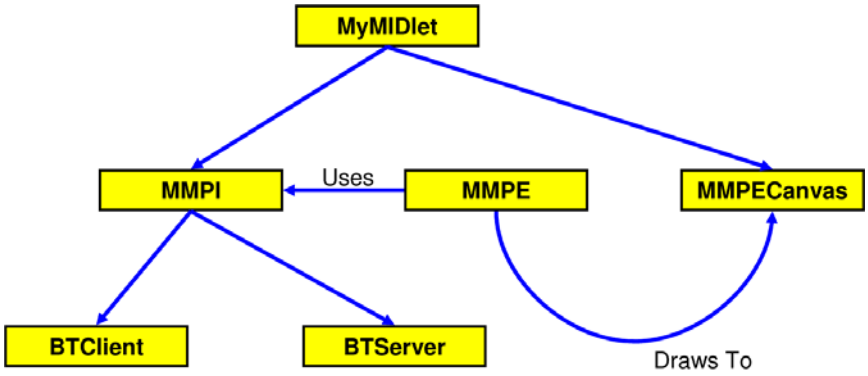

Fig. 1. MIDlet Structure Using the MMPE Library.

In standard Client / Server architectures the Server is the first system to be executed as it must establish what services it has to offer clients that connect to it. This role is reversed in Bluetooth communications. The creation of a devices to act as a client requires the client to create a Server connection. This connection allows the device to advertise that it as active. The Master device in the Bluetooth system will then carry out device and service discovery to find all the devices that have applicable services to offer. To establish connections to all the relevant devices found the Master must establish a client connection to each one, thus forming a star network topology. This is somewhat akin to the SETI project where it is the client applications that carry out the processing, and simply return the results to the server.

One of the key advantages of the MMPI library is that the programmer no longer has to develop the same repetitive code each time they wish to develop a Bluetooth based application. This includes the creating of Client and Server connections, device and service discovery and

the handling of communications streams such as DataInputStream and DataOutputStream. The programmer is completely abstracted from having to develop any Bluetooth specific code. To allow a program to begin communication between other devices all that is necessary is to instantiate and MMPI object on each device. This alone removes allows the programmer to write one single line of code as opposed to several hundred, just the initialise the and find all the devices that register themselves as an "mmpiNode".

On creation of an MMPI world the use of the methods getSize() and getRank() allows the devices in the system to establish the number of devices in the network, and allows each device to get its particular identifier in the form of a integer from zero (the root node) through to $n-1$. To achieve communication any one of the point to point or global communication methods may be called, as with MPI each method takes a device identifier as a parameter, be it representing the device to send to, or the device to receive from. To achieve a broadcasting of data throughout all the devices in the world a total of just five method calls are required. These include the creation of the world, the establishment of the world size and the rank of each device, a call to the broadcast method and finally a call to the finalise method to destroy the world.

2.1 World Creation

The establishment of the fully interconnected mesh network is one of the most time consuming and complex tasks of the MMPI system. All devices that are started up as clients establish server connections to advertise their presence, this is a blocking operation. To create a world with five nodes four of them must be initialised in this fashion. The final node to be started up is the root node of the system, and is responsible for the searching and establishing of connections to the other devices. This process occurs in a number of distinct steps. Firstly device and service discovery is carried out to find all devices that are registered as an "mmpiNode". The discovery process can take quite a significant amount of time, with the Bluetooth standard specifying 10.24 seconds for the establishment of the device discovery process alone. In reality it often requires several more second to fully complete the device discover process. Testing the world creation times gave times of 12,019ms and 12,926ms for the discovery of two and three devices respectively. Service discovery required 6,109ms for three devices and 4,367ms for two. The creation of the mesh required 2,135ms in the case of three devices and 1,251ms for two devices. The total time to create a world with three device was over 21 seconds taking into account the discovery process (over 19 seconds) and the mesh creation stage. Once this has been established a series of DataInputStreams and DataOutputStreams are created to allow for channels of communication between the root node and all the other devices within the system. The world now has a star network topology.

To allow for the seamless communication of one device with any other device the network must be transformed into a mesh structure. This requires the establishment of another series of Client and Server connections. All of the communication streams are stored in an array on each device. In the case of a world with five devices each node will have an array of size five, however the only element of the array that doesn't hold a communications stream is the index that refers to the identifier of the current device. Device two for example will have two streams stored at elements zero and one. Element two has no stream as it refers to itself, and the remaining elements again contain communication streams. On creation of the star

network the root node has client connection streams spanning across all elements of the array that are greater than the device identifier (zero). Each of the client devices contain just one stream (a server connection) all laid down at element zero on each of their respective arrays.

The creation of the mesh is established by all clients who's remaining elements are less than the identifier of the device to create client based connection streams. All elements of the stream array who's id is higher than the device identifier will establish Server connections. The establishment of the Server and Client connections must be done in a synchronised manner to ensure that each device identifier refers to the exact same device throughout the system. As each Server connection is established it is communicated back to the root node which then forwards on the message to the corresponding client which can then create a Client connection to establish two way communications with Server object just created. Under this system a Client device will establish Server connections to all the other Client devices who's id is higher than it. The end result of this entire process is the establishment of an array of communication streams of each device that allow for communication with every other device within the world (Figure 2).

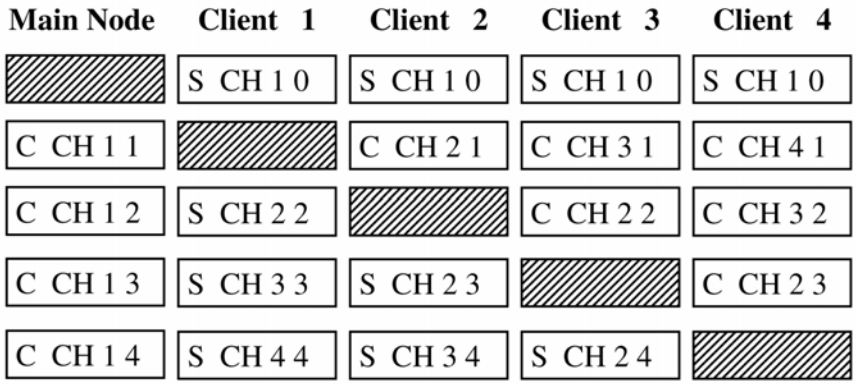

Fig. 2. Communications Interconnect.

2.2 Parallel Graphics

The Multi-Processing Environment (MPE) is an extension to the standard set of MPI functions. MPE includes a set of profiling libraries, utility programs, and a set of graphical visualisation tools. The MPE graphic routines allow for the display of simple graphics with an X Window System. There are also routines for getting mouse input, both selection of a point and selection of a region via dragging. As seen in Figure 1 two additional classes are use to allow for mobile parallel graphics, that of MMPE and MMPECanvas to where the actual drawing takes place.

The MPE graphics routines allow for the drawing of points, lines, rectangles and circles. One can also draw a filled rectangle. With the MMPE implementation the developer has a complementary set of both drawing and filling methods. Including several drawing operations not directly implemented through J2ME itself. The complete set of drawing methods may be seen in Listing 3. These methods are the root node drawing methods where all the graphics are displayed on the root note. An identical set of methods exist to allow for the broadcast

drawing of the graphic primitives across all devices. All of these method names have "Bcast" appended to the method name for example drawRect(...) and drawRectBcast(...). Of all the drawing primitives the drawImage(...) is the most expensive in terms of communication due to the transmission of pixel information.

```
public void setMPIWorld(MMPI commNode)
public void setGraphicsHandle(Graphics gr)
public void drawPoint(...)
public void drawRect(...)
public void fillRect(...)
public void drawLine(...)
public void drawCircle(...)
public void fillCircle(...)
public void drawArc(...)
public void fillArc(...)
public void drawRoundRect(...)
public void fillRoundRect(...)
public void fillTriangle(...)
public void drawTriangle(...)
public void drawString(...)
public void drawSubString(...)
public void drawImage(...)
```

Fig. 3. MMPE Drawing Methods.

Figures 5 & 4 clearly depict how the graphical primitives are drawn on screen in both the root drawing mode and the broadcast drawing mode. Two non drawing methods also exist within the MMPE class, setMPIWorld(...) is used to allow the MMPE system to obtain a reference to the MMPI communications system, similarly setGraphicsHandle...) is used to obtain a reference to an instantiated MMPECanvas object.

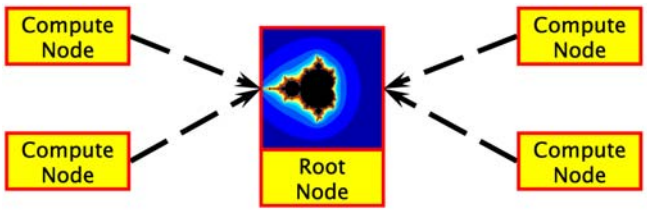

Fig. 4. Standard MPE / MMPE Graphics Drawing to the Root Node.

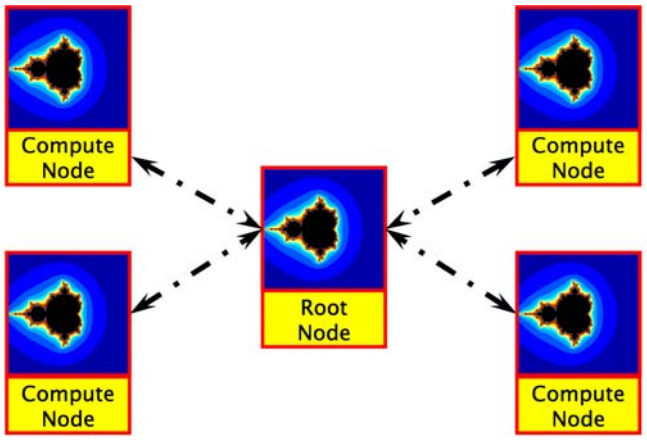

Fig. 5. Broadcast MMPE Graphics Drawing to All Deivces.

2.3 Fault Tolerance

In the case of any parallel machine, the addition of more hardware components results in the increased possibility of a fault occurring. In general users of parallel machines want their programs to adapt to faults and continue execution. In many cases faults may be detected and the system reconfigured to allow for same, however the one can never achieve a truly fault tolerant system as their is always the possibility of a complete system wide failure.

The Message Passing Interface in itself is simply a standard that specifies how a correctly written parallel program may be achieved. Many say that MPI is not fault tolerant, but this is neither true or false. Most believe that when a process dies then all MPI nodes that are part of the world should so too die, this however is not the case. The default operation for when a process becomes unavailable is indeed to kill all remaining processes in the world, this is achieved through the built in MPI_ERRORS_ARE_FATAL error handler. Therefore if the developer carries out no error handling in respect to this type of error, then indeed all other nodes in the world will die before the process should normally terminate by calling the MPI_Finalize function. This default behaviour for all nodes dying on a node having an error was decided by the MPI Forum to be the most useful default behaviour.

Fault tolerance is not an actual property of MPI itself. In general it is assumed that that the parallel program will execute on reliable hardware. How hardware faults are handled are implementation specific. Gropp and Lusk [23] argue that fault tolerance is a property of both an MPI program coupled with and MPI implementation. In this case they consider the MPI implementation to be both the hardware and software environment on which an MPI application may execute.

In general a fault tolerant program and underlying infrastructure should be capable of surviving failures such as system crashes and network failures. At the highest level the MPI program should be capable of automatically from recovering from a set of faults without any change to the apparent behaviour of the program. The next level is that notifications of failures should be posted to the MPI program so appropriate action may be undertaken. At the third level, certain operations may become invalid, for example failure of a node may

rule out the possibility of collective communication routines, but standard point to point communication may still continue between unaffected nodes. The fourth level makes use of checkpointing thereby allowing a program to save its state to persistent storage, abort and restarted from the checkpoint. The final level of survival may use a combination of the previous approaches.

Most approaches to fault tolerance have a set of three distinct requirements: detection of failures, the maintenance of state information to continue the computation and the ability to restart.

Any implementation conforming to the MPI standard is responsible for detecting and handling network faults, this may include message retransmission or the informing of the application through the use of an error code. Essentially the contents of a message transmitted from one node should be identical to the message received on another node.

Several fault tolerant MPI implementations are currently in existence. MPICH-V [24] is considered to be one of the most complete featuring checkpointing and message logs to allow aborted processes to be replaced. This implementation comes at a cost of approximately doubling the communication times, for the provision of full recovery. The LAM based MPI-FT [25] also uses a similar approach to fault tolerance while FT-MPI [26] modifies some of the standard MPI semantics.

Fagg and Dongerra [26] discuss four possibilities for the recovery of a communicator that has an error state. To reconstitute a valid communicator it is necessary to rebuild it using modified versions of one of the MPI communicator build functions such as MPI Comm create, MPI Comm split or MPI Comm dup. SHRINK allows the communicator to be reduced in size so that the structure is contiguous, the requires the modification of the ranks. BLANK is similar to shrink except the communicator now contains blanks instead of references to nodes that are now unavailable. Communications with a gap will result in errors, therefore prohibiting the effective use of global communication functions. REBUILD is the most complex mode available that forces the creation of new processes to fill any gaps until the communicator is fully reconstituted. This mechanism can either fill gaps in the communicator or shrink the communicator and add additional nodes so it returns to its original size. The final option is ABORT which will result in the graceful abortion of the application of detection of an error.

2.4 The Mobile Environment

Achieving a reasonably reliable parallel system using specialised hardware and cabled infrastructure is one thing, but achieving a degree of fault tolerance in a mobile environment with general use hardware is quite a different challenge. Firstly the communications medium, is far less reliable, and is more prone to interference. Nodes in a standard parallel machine are situated in a fixed geographic location interconnected with high speed cabling. In the mobile environment nodes may vary in geographic location, as a result lines of communication between some of the nodes have the possibility of being sporadic. Some of the possibilities for failure in a mobile environment include nodes being terminated by a user, nodes terminating for an exceptional reason, nodes leaving the Bluetooth range, nodes reentering the Bluetooth range, environmental conditions effecting the wireless communication streams.

The level of fault tolerance required can vary from application to application, the MMPI library has proved very successful in the area of multiplayer gaming and mLearning, greatly

reducing the development time, and reducing code complexity. As mentioned by Fagg and Dongerra one can generally use one of four possibilities: SHRINK, BLANK, REBUILD or ABORT. In the case of a multiplayer game of more than two players the loss of one or more nodes would in general have no great effect on the game itself. One a node has been detected as being no longer viable, the avatar representing that node may be removed from the game or replaced by an inactive on screen avatar representing the lost player. This represents the simplest means of handling errors. In the case that a communication with another node results in the throwing of a null pointer exception on a DataInputStream / DataOutputStream object, one can consider that the node in question is no longer available, and hence the reference to those streams can be nullified in the simplest case. Each node of an MMPI world maintains an array, both of Input and Output streams, should one wish to shrink the world on the event of an error then the array references of each node would accordingly need to be reduced. This requires the communication of the ID of the lost node to the remaining devices and the corresponding removal of that element from the array. In the case of a world with just two devices partaking then the simplest procedure is to Abort the World.

The mobile environment can find itself in a state of flux, and not only may devices become detached and lost, but other devices may become active during the computation period of an application. In the case of a mobile application carrying out a parallel processing task, one may wish to try and rebuild the world, thus any new devices that have entered the Bluetooth communications range may be detected. This however would require the execution of the device discovery, and service discovery phases once more, which can take in some cases almost twenty seconds to complete. Rebuilding the World could be quite practical in some applications especially mLearning where there may be a long delay between message transmission. To ensure that a World could be reestablished with little or no knowledge of the user, it is necessary to check for communication errors on a regular basis. One simple means of doing this is to carry out a short burst of global communication in the background. The reduce() method can prove quite effective for this. The alternative to this is to checkpoint the application, either at regular intervals or at key points within the applications life-cycle. Two possibilities exist for the storage of checkpoints, the first is to use the devices filesystem, by use of the JSR 75 API, the other is to use the MIDlets Record Management System (RMS) database. Unless a MIDlet has been signed then the use of the filesystem can be problematic, as user authorisation is constantly required both for navigation through the directory structure and for file access. One may also find that some devices, while supporting Bluetooth (JSR 82) do not support JSR 75, the Nokia 6600 being just one example. In light of this the RMS may prove to be a more compatible alternative. The storage capacity may be far more limited, therefore one may store only a few checkpoints in the case where a significant amount of state information is necessary to restore the application back to a similar state before the detection of the fault.

2.5 Cross Platform Usage

The development of Bluetooth enabled applications designed for a phone is generally straight forward, as the virtual machines of most modern day phones support the JSR 82 API. Cross platform Bluetooth development is far more complex, as one must deal with both inconsistencies of both hardware and software implementations. Several Java based libraries currently

exist for Bluetooth development, some being freely available while others require significant investment to acquire the API's and associated tools. Cross platform development is thus, not as straight forward as one would expect, as one often faces strange and unusual errors either at compile time or runtime. Java ME has been around for some years now, but one significant problem with it is fragmentation. One can not simply develop a Java program that can be written once and is guaranteed to run anywhere. Java ME is essentially an amalgam of a whole host of JSR's. The majority of devices support only a small subset of these API's. Therefore when one relies of such API's that are used only my a limited set of devices, or other third party API's one is immediately limiting the range of devices upon which one can expect an application to successfully execute.

This should become far less of a problem in the near to medium future especially as Sun Microsystems' [27] has announced (October, 2007) that Java SE will gradually supplant Java ME. Mobile devices are becoming more and more powerful, and as such they are now on the verge of being capable of running Java SE applications. Perhaps a few years from now, one will truly be able to "write once, run anywhere".

3 Mobile Multimedia Applications

The number of Bluetooth enabled multiuser / device applications that may be developed using the library are limited only by one's imagination. This section discusses just a few of the possible application areas to which the library may be applied. Wireless communications technology such as Bluetooth is finding it's way into a myriad of devices, be it from the kitchen to the living room or even the automobile industry. Therefore inter-process / device communication will become all the more common place in the short to medium future.

3.1 Combined Processing Power

This was the initial reason for development of the library, to harness the computing power of several devices, to reduce the overall computation time of processor intensive tasks.

The classical approach to matrix multiplication is an $O(n^3)$ problem. The best possible complexity is $O(n^2)$ as all n^2 cells must be examined. Several improvements on the $O(n^3)$ complexity are viable including Strassen's algorithm $O(n^{2.807})$ [28], and better yet the Coppersmith-Winograd algorithm $O(n^{2.376})$ [29].

For the parallel computation we consider that the matrices are square $A, B, C \in M_n(R)$ and the number of processors or mobile devices to use in computation is p. The matrices A and C are split into p similar $\frac{n}{p} \times n$ sub-matrices as follows

$$\begin{pmatrix} C_0 \\ C_1 \\ ... \\ C_{p-1} \end{pmatrix} = \begin{pmatrix} A_0 \\ A_1 \\ ... \\ A_{p-1} \end{pmatrix} \cdot B = \begin{pmatrix} A_0 \cdot B \\ A_1 \cdot B \\ ... \\ A_{p-1} \cdot B \end{pmatrix}. \tag{1}$$

This partitions the computation of $C = A \cdot B$ into p computation

$$C_i = A_i \cdot B, i = 0, 1, ..., p-1, \tag{2}$$

where each processor/mobile device i computes a smaller product. To accommodate this the matrix A scattered evenly among all the nodes in the system. This is followed by the

broadcasting of the matrix B. Once this communication has been completed each device can compute its own section of the matrix my calling the product(...) method (Algorithm 1).

```
create MMPI World;
get Node ID (Rank);
get communicator size;
if (RANK == MASTER) then
    initialise matrices;
end
scatter(A);
broadcast(B);
compute Sub Matrix Product;
gather(C);
if (RANK != MASTER) then
    output results;
end
```

Algorithm 1: Matrix Multiplication

The scattering and broadcasting of entire matrices involves significant communications costs. The generation of Fractal images is far more suited to mobile parallel computation as the initial amount of communication is very small. The transmission of the final generated image segments does require a substantial amount of transmission time, typically a few seconds, but this is of minor consequence when compared to processing times of hundreds of seconds when executed on a single device. The generation of the Mandelbrot Set (Algorithm 2) is a classical fractal generation problem often referred to as an "embarrassingly parallel computation" as the image area may be divided into distinct parts without reliance on data from any other sections. The MSet may be obtained by iterating the function $f(z) = z^2 + c$. Many other Mandel like sets may be produced when the generation function has the form of $f(z) = z^u + c^v$.

```
for each (x,y) in [x_min, x_max] × [y_min, y_max] do
    c = x + i × y;
    find the orbit of z_0 while under the threshold R;
    if (all the orbit points are not under the threshold) then
        draw (x,y);
    end
end
```

Algorithm 2: Mandelbrot Set Algorithm

The essential parameters that need only be transmitted to initialise the fractal generation process are the $xmin, ymin, xmax, ymax$ coordinates of the complex plain that represents the required imaging area to be computed. Several methods of load balancing exist including: Uniform Block, Cyclic and Dynamic methods. The Uniform block approach allows the i^{th} node to receive image areas of the form $i \times w/nrNodes, i \times w/nrNodes + 1, ..., (i+1) \times w/nrNodes - 1$ for processing. Cyclic decomposition uses smaller strips $\{S_0, S_1, ..., S_{p-1}\}$. Partitioning of p strips onto the nodes is performed in a cyclic matter, so node i would receive the strips $\{S_{i+j \times nrNodes} : \ j = 0, 1, ..., \frac{p}{nrNodes} - 1\}$. Dynamic balancing divide the imaging

plane into a grid of cells and distributes these cells out for processing, can each is returned another cell for processing is given out until all cells in the work pool have been exhausted.

4 Mobile Learning

The emergence of mLearning has resulted from the advancement in wireless technology in recent times. This coupled with the increased mobility of e-Learners have given rise to a potential revolution in education, that of mLearning. It is probably suffice to say that the students of today be they at University, Secondary or even Primary level all have access to mobile technology. Many at the secondary and University level sport 3G phones that are Bluetooth enabled and have JVM's with support for a variety of advanced packages from Scalable Vector Graphics (SVG) to Mobile 3D Graphics and Bluetooth connectivity. With such a huge technology base in existence, it is time that these devices were put to an education use. In many a recent year the first words to be heard at the start of any class or lecture is nearly always "please turn off your mobile phones". The provision of education mLearning tools means that students can learn in new and innovative ways using a platform that they use daily and have a thorough mastery of its use. Many of the mLearning materials to date have relied on the Wireless Application Protocol (WAP) to provide resources such as dictionaries and revision flash cards through the use of the Wireless Markup Language (WML). Clearly there is a call for more interactive means of mLearning, where student should take an active rather than passive interaction modality. It has long been known that people best learn by seeing, hearing and more importantly by doing. Using mLearning to provide interactive games and quizzes where by a teacher can monitor the progress of the students, is one fun way to get students to learn more, by doing so with a device they use and enjoy so much.

A useful application of MMPI with mLearning is to provide an interactive tool that allows a teacher to generate questions which can be sent to the students phones [30]. The students should then work out the answers and send the results back to the teacher. This general paradigm is especially useful for subjects such as mathematics, where sometimes the students may be intimidated by complex concepts. Providing a fun and interactive means of working with such subjects may help to improve students interest and overall understanding of the subject. The subject area of mathematics is very suitable as it is a simple matter to generate new questions for the students to answer, example areas include Algebraic Linear Simultaneous and Quadratic equations. These applications take the form of a Master / Client relationship, where the teachers phone or PDA generates the questions, and broadcasts same to all the student (Client) phones. The student them must solve the problem presented by hand and enter their results on the phone and transmit it back to the teacher. On receipt of all the results the teacher can instantly see how many students in the class got the question correct. A response is also sent back to each individual students phone indicating if they got the question correct or not (Algorithm 3). This system allows the teacher to quickly and accurately gauge the competency of the students grasp of the topic area.

A group of secondary school students tested the application, whom for all this was their first experience of an mLearning application. The overall results proved promising with 89% enjoying the new means of classroom interaction that deviated from the standard classroom based learning environment. Over three quarters 76% of the students responded that they would like to see similar applications incorporated into their future classroom schedule.

```
repeat
    if (RANK is MASTER) then
        Generate questions;
        Send to non MASTER devices;
        for (each non MASTER device) do
            Acquire result from non MASTER;
            Validate results;
            Transmit result to non MASTER;
        end
    end
    if (RANK not MASTER) then
        Acquire question from MASTER;
        Acquire solution from User;
        Transmit solution to MASTER;
        Acquire validated result;
    end
until while active ;
```

Algorithm 3: mLearning Student / Teacher Algorithm

5 Multi-player Gaming

Games themselves are usually classified into several genres that reflect the type of game, such examples include: Strategy, Puzzle, Role Playing, Action, First Person Shooter and Driving. Wang et. al [31] proposed a new classification focused on multi-player mobile gaming. This may be divided into two distinct categories: Updating and User Interaction. Updating is subdivided into: U1 Asynchronous, Synchronous and Real-time. The user interaction dimension is defined as: Controlled, User Interaction, Automatic Triggered and Automatic. In summary this work says that games falling into the categories of Real-time, Automatic Triggered and Automatic are not well supported for mobile devices.

Originally the MMPI library was developed to allow for parallel computation, but as it provides a very simple means to allow cross device communication it has been successfully employed to the area of multi-player mobile gaming.

Often with many games the scene needs to be updated as the user provides input from the keypad, be it moving an avatar or firing a weapon. Under J2ME such interaction is handled by the keyPressed(...) method, therefore it is necessary to transmit the updated coordinates to the other devices. To receive a simple Thread is sufficient just to pair up the sends with receives and update the required elements of the receiving device. In games that have random moving elements, the generation of the random values should be carried out on the root node and the updated values transmitted to all connected devices to ensure the random elements are displayed in sync with all of the devices within the world.

An alternative and more elegant means of multi player game communication is continually acquire user input and broadcast same across the world (Algorithm 4).

Figure **??** shows an example of a maze game in operation. The keys (generated by the root node) are placed at random locations with their position updated every second. Each player is in control of a different avatar, the objective being to find the way around the maze, collecting keys to open the doors, and get to the treasure chest to clear the level.

```
create MMPI World;
get Rank & Size;
repeat
    Draw Background Scene;
    Acquire User Input;
    for (all nodes) do
        broadcast Moves;
        update Character Positions;
    end
until while active ;
```

Algorithm 4: MMPI Game Broadcasting

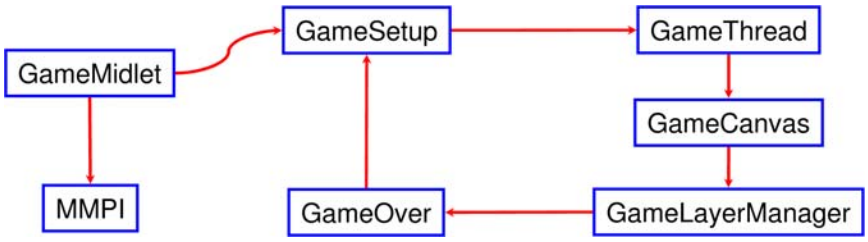

Fig. 6. Template for MMPI Gaming.

5.1 Communications

Interactive realtime multi-player mobile games typically require the regular transmission and receipt of small packets of data to ensure cross device synchronisation of the on screen game. Often the amount of data to be transmitted is just a few bytes, such as avatar coordinates. The transmission of blocks of data directly form one device to another showed that 2,500 bytes of data could be transmitted in 47ms. Transmission time reduced to 15ms for 200 bytes of data. Clearly on can maintain a high on screen frame rate with such communication times. In another experiment data was transmitted to two devices, and an acknowledgement sent back from each, such replies are generally unnecessary for gaming, but the communication times still prove quite positive requiring 32ms to transmit 40 bytes of data to two devices and receiving an acknowledgment from each. Even the transmission of 200 bytes yielding a round trip time of 63ms still allows a reasonable frame rate to be maintained.

6 Conclusion

The MMPI library has been shown to aid and simplify the development of Bluetooth based applications. Several example application areas where it may be used have been demonstrated. Incorporating the MMPI library with a simple to use framework one can quickly transform a single player game into a multi-player game. It has been shown that games that require real time updating to ensure all users screens are synchronised is perfectly viable using the library. Within the realm of mLearning it provides an easy to use mechanism to develop educational based application that can help to provide new and interactive methods of in class teaching. It is clear that the library provides an essential mechanism for effective Bluetooth enabled application development. The library also contains an addition class (MMPE) that allows for

the drawing of graphic primitives across all nodes and also just to the root node, allowing for the potential to solve may parallel graphics problems within the mobile environment.

The area of fault tolerance within the mobile environment poses many new possibilities for the chances of an error occurring. The actions that may be taken in the event of an error being detected can vary depending on the type of application. In some cases it is simplest just to carry on, while in other applications that have little communication one may have sufficient time to rebuild a world with little or no knowledge of same required by the user. Checkpointing of minimal state using an RMS database can prove to be effective in parallel computation scenarios.

Acknowledgements

This research project was funded under the "Irish Research Council for Science, Engineering and Technology" funded by the "National Development Plan".

References

1. N. Gohring (2006), *Mobile phone Sales Topped 800 Million in 2005, says IDC*, http://www.computerworld.com/mobiletopics/mobile/story/0,10801,108146,00.html?source=x10.
2. COMREG (2007), *Quarterly Key Data Report March 2007*, http://www.comreg.ie/_fileupload/publications/ComReg0717R.pdf.
3. S. Freak (2005), *Nokia 6680 is Loosing the Battle to 6630*, http://www.symbian-freak.com/news/0305/6680.htm.
4. JBenchmark (2007), *Nokia 6630 Performance Details*, http://www.jbenchmark.com/phonedetails.jsp?benchmark=v1&D=Nokia%206630web_JBenchmark_Nokia3220.pdfweb_JBenchmark_Nokia3220.pdf.
5. ARM, *ARM Cortex-A8* (2005), http://www.arm.com/products/CPUs/ARM_Cortex-A8.html.
6. ARM (2005), *ARM Introduces Industry's Fastest Processor for Low-Power Mobile and Consumer Applications*, http://www.arm.com/news/10548.html.
7. Apple Computers (2007), *Apple iphone*, http://www.apple.com/iphone/.
8. D. M. Mountain (2007),*Geographic Information Retrieval in a Mobile Environment: Evaluating the Needs of Mobile Individuals*, Journal of Information Science.
9. D. M. Mountain (2005), *Exploring Mobile Trajectories: An Investigation of Individual Spatial Behaviour and Geographic Filters for Information Retrieval*,PhD thesis, School of Informatics, City University, London.
10. S. Stachan (2007), *Multimodal, Embodied and lacation-Aware Interaction*, PhD thesis, National University of Ireland, Maynooth.
11. Bluetooth.com (2007), *The Official Bluetooth Website*, http://www.bluetooth.com/.
12. Bluetooth.org (2007), *The Official Bluetooth Membership Site*, http://www.bluetooth.org/.
13. M. Gong, S. Midkiff, and R. Buehrer (2003), *A new Piconet Formation Algorithm for UWB ad hoc Networks*, IEEE Conference on Ultra Wideband Systems and Technologies, pp. 180–184.
14. *Top 500 Supercomputer Sites* (2007), http://www.top500.org/.
15. *Seti@Home* (2007), http://setiathome.ssl.berkeley.edu/.
16. *Berkeley Open Infrastructure for Network Computing* (2007), http://boinc.berkeley.edu/.
17. MPI (2007), *The Message Passing Interface (MPI) Standard*.
18. MPICH (2007), *MPICH - Free Implementation of MPI*.
19. HPJava (2007), *mpiJava*, http://www.hpjava.org/mpiJava.html.
20. DSG (2007), *Message Passing in Java (MPJ) Project*, http://dsg.port.ac.uk/projects/mpj/.
21. D. C. Doolan, S. Tabirca, and L. T. Yang (2006), *A Bluetooth MPI Framework for Collaborative Computer Graphics*, International Symposium on Parallel and Distributed Processing and Applications (SIPA 2006), Sorrento, Italy, pp. 561–572.

22. D. C. Doolan, S. Tabirca, and L. T. Yang (2006), *Mobile Parallel Computing*, 5th International Symposium on Parallel and Distributed Computing (ISPDC06), Timisoara, Romania, pp. 161–167.
23. W. Gropp and E. Lusk (2002), *Fault Tolerance in MPI Programs*,Cluster Computing and Grid Systems Conference, `http://www-unix.mcs.anl.gov/~gropp/bib/papers/2002/mpi-fault.pdf`
24. G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov (2002), *MPICHV: Toward a Scalable Fault Tolerant MPI for Volatile Nodes*, Supercomputing, ACM/IEEE 2002 Conference.
25. S. Louca, N. Neophytou, A. Lachanas, and P. Evripidou (2000), *Mpi-ft: Portable fault tolerance scheme for mpi*, Parallel Processing Letters, Vol. 10(4), pp. 371-382.
26. G. E. Fagg and J. J. Dongarra (2000), *FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World*, Lecture Notes in Computer Science, Vol. 1908, pp. 346-353.
27. S. Shankland (2007), *Sun Starts bidding Adieu to Mobile-Specific Java*, `http://www.news.com/8301-13580_3-9800679-39.html?part=rss&subj=news&tag=2547-1_3-0-20`
28. V. Strassen (1969), *Gaussian Elimination is not Optimal*, Numer. Math, Vol. 13, pp. 354–356.
29. D. Coppersmith and S. Winograd (1990), *Matrix Multiplication via Arithmetic Progressions*, Journal of Symbolic Computation, Vol. 9, pp. 251–280.
30. D. C. Doolan and S. Tabirca (2006), *MMPI making Maths in the Classroom Fun*, Information an International Interdisciplinary Journal, Vol. 9(6), pp. 807–812.
31. A. I. Wand, M. S. Norum, and C.-H. W. Lund (2006), *Issues related to Development of Wireless Peer-to-Peer Games in J2ME*, International Conference on Internet and Web Applications and Services (ICIW 20006), pp. 115–120.