WorkPackage WP11: Enhanced Reality Live Role-playing

Deliverable D11.9: Second generation core platform.

Pär Hansson, SICS
Karl-Petter Åkesson, SICS
Anders Wallberg, SICS

Release date: 21 February, 2007
Status: Public
Executive Summary

This software deliverable includes the core system for the eLARP showcase second phase game Momentum. The system is built on top of the PIMP platform which has been delivered as a project internal deliverable in D7.5. A public version of PIMP will be provided in the D7.6 but this deliverable includes a brief explanation of PIMP to give a better understanding of the game system.

This deliverable can be of interest to all IPerG partners and others developing games including sensors and actuators. It can also be of interest for someone creating quick prototypes of some game mechanism based upon sensor and actuator information. This deliverable covers the limited set of sensors and actuators used within the Momentum game. Furthermore, one could quite easy create simple prototypes of other pervasive services based on a Momentum-like game system though the functional objects in this release are much targeted for games. Anyone interested in understanding the structure of game systems is a potential target for this deliverable, which also to some extent explains the different interactive artefacts within the game.

NOTE: The core game technology's heavy dependency on specialized hardware means this software deliverable does not intend to make it possible for anyone to set up and run the system, especially not the way it was used during the Momentum game which took place in October of 2006.
## Deliverable Identification Sheet

<table>
<thead>
<tr>
<th>IST Project No.</th>
<th>FP6 – 004457</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acronym</td>
<td>IPerG</td>
</tr>
<tr>
<td>Full title</td>
<td>Integrated Project on Pervasive Gaming</td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://www.pervasive-gaming.org/">http://www.pervasive-gaming.org/</a></td>
</tr>
<tr>
<td>EU Project Officer</td>
<td>Albert GAUTHIER</td>
</tr>
</tbody>
</table>

### Deliverable

- **D11.9** Second generation core platform

### Work package

- WP11 Enhanced Reality Live Role-playing

### Date of delivery

- **Contractual** M 29
- **Actual** 31-Mar-07

### Status

- version. 1.0
- final

### Nature

- Prototype
- Report
- Dissemination

### Dissemination Level

- Public
- Consortium

### Authors (Partner)

- Pär Hansson(SICS), Karl-Petter Åkesson(SICS), Anders Wallberg(SICS)

### Responsible Author

- Pär Hansson

### Email

- par@sics.se

### Partner

- SICS

### Phone

- +46 8 633 1575

### Abstract (for dissemination)

Ubicomp software platform for large-scale long-term pervasive live action role-playing game.

### Keywords

Ubicomp distributed live action role-playing game system

### Version Log

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>07-02-21</td>
<td>001</td>
<td>Pär Hansson</td>
<td>Initial version ready for review.</td>
</tr>
<tr>
<td>07-04-05</td>
<td>002</td>
<td>Pär Hansson</td>
<td>1st reviewer’s comments addressed.</td>
</tr>
<tr>
<td>07-04-10</td>
<td>003</td>
<td>Pär Hansson</td>
<td>2nd reviewer’s comments addressed.</td>
</tr>
</tbody>
</table>
# Table of Contents

**Table of Contents**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>3</td>
</tr>
<tr>
<td>1 Overview</td>
<td>5</td>
</tr>
<tr>
<td>2 The Momentum Game</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Gameplay overview</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Technological Realization</td>
<td>7</td>
</tr>
<tr>
<td>3 Important Game Artefacts</td>
<td>8</td>
</tr>
<tr>
<td>3.1 Urim Bracelet - a link to the spirit realm</td>
<td>9</td>
</tr>
<tr>
<td>3.2 Stele of Unity – the locator</td>
<td>11</td>
</tr>
<tr>
<td>3.3 Elemental Tablet - The Input device</td>
<td>12</td>
</tr>
<tr>
<td>3.4 Thumin – the identifier</td>
<td>13</td>
</tr>
<tr>
<td>4 Core Momentum Software</td>
<td>14</td>
</tr>
<tr>
<td>4.1 Phone Client</td>
<td>14</td>
</tr>
<tr>
<td>4.2 Game Server</td>
<td>15</td>
</tr>
<tr>
<td>4.2.1 Objects used by the game server</td>
<td>15</td>
</tr>
<tr>
<td>4.2.2 Game Rules</td>
<td>16</td>
</tr>
<tr>
<td>4.2.3 Game Master Intervention</td>
<td>17</td>
</tr>
<tr>
<td>4.3 Starter</td>
<td>18</td>
</tr>
<tr>
<td>5 Discussion</td>
<td>18</td>
</tr>
<tr>
<td>5.1 Reconfigurability</td>
<td>18</td>
</tr>
<tr>
<td>5.2 Hard coded game rules</td>
<td>18</td>
</tr>
<tr>
<td>5.3 Debugging tools</td>
<td>19</td>
</tr>
<tr>
<td>5.4 The use of Bluetooth</td>
<td>19</td>
</tr>
<tr>
<td>5.5 System complexity</td>
<td>20</td>
</tr>
<tr>
<td>5.6 Managing resources</td>
<td>21</td>
</tr>
<tr>
<td>6 PIMP – Pervasive Interactive Mobile Platform</td>
<td>21</td>
</tr>
<tr>
<td>6.1 PIMP concept</td>
<td>21</td>
</tr>
<tr>
<td>6.2 Discussion</td>
<td>23</td>
</tr>
<tr>
<td>6.2.1 Status</td>
<td>23</td>
</tr>
<tr>
<td>6.2.2 Strengths</td>
<td>23</td>
</tr>
<tr>
<td>6.2.3 Weaknesses</td>
<td>23</td>
</tr>
<tr>
<td>6.2.4 Goals and plans</td>
<td>23</td>
</tr>
<tr>
<td>7 Other Software Components</td>
<td>24</td>
</tr>
</tbody>
</table>
8Orchestration tools.................................................................................................................. 25

9Setting up and running the Momentum game system........................................................... 25

9.1 Session server..................................................................................................................... 26

9.1.1 Game Server.................................................................................................................. 27

9.1.1.1 Command line arguments......................................................................................... 27

9.1.2 Starter............................................................................................................................ 27

9.1.2.1 Command line arguments......................................................................................... 27

9.1.2.2 Starter property file................................................................................................... 27

9.1.2.3 A phone connects...................................................................................................... 27

9.1.2.4 Start sequence........................................................................................................... 27

9.2 Simple text logger.............................................................................................................. 28

9.3 SQL logger........................................................................................................................ 28

9.4 Servlet interface............................................................................................................... 29

9.4.1 Requirements................................................................................................................ 29

9.4.2 Setup............................................................................................................................ 29

9.4.3 Starting.......................................................................................................................... 30

9.5 Phone client...................................................................................................................... 30

9.6 PocketPC........................................................................................................................... 32

9.6.1 Runtime Installation..................................................................................................... 33

9.6.2 Bluetooth support......................................................................................................... 33

9.6.2.1 Get a working version.............................................................................................. 34

9.6.2.2 Installation............................................................................................................... 34

9.6.3 Software installation...................................................................................................... 34

9.6.4 Shortcut setup.............................................................................................................. 34

9.6.5 Starting the application............................................................................................... 35

10Appendix A.......................................................................................................................... 36
ABSTRACT

This document and accompanying web site includes a young version of the different components that make up the technical platform upon which the second eLarp game, Prosopopeia Momentum, is realized. It describes the technical functionality of components, how these are used within the game. Furthermore, we explain how to use these components from a game designers perspective. Thus, by reading this document it should be possible for someone else to get an idea of how to set up another game based upon this technical platform. NOTE: intention of this deliverable has changed since original plan. Since much of the game play for Momentum is based on specific hardware, media etc it is not our intention that anyone should be able to set up a runnable system using this document and software. These hardware components have been delivered in the project internal deliverable D7.5 and will be publicly available in the forth coming D7.6. This document also includes a brief discussion on the viability of this type of a system.

1 OVERVIEW

This deliverable is the first version of the game system being developed for the second game developed by the eLarp showcase of IPerG. The system is built on top of the Pervasive Interactive Mobile Platform (PIMP) developed within IPerG. This platform is currently only delivered internally within IPerG but a short description is provided here to get an understanding of the whole game system.

The game is called Momentum and is based in the same fictional world as the previous Prosopopeia game, see deliverable D11.3 (and http://iperg.sics.se/iperg_gamesub5.html) for more details. In this game world technology can provide a bridge to the world of the dead. In Momentum the game play is much more focused on both player versus player (PvP) game elements as well as player versus environment (PvE). More traditional game elements are introduced, the PvP game is very similar to existing first person shooters or games like Laser Quest. The PvE game is about area control. More information about the actual game play will be described in upcoming deliverables.

The system is constructed around one mobile phone per player, a central server and a number of game artefacts. The mobile phone is not really used as a game artefact as such but merely works as a communication node. Through the phone, with the use of Bluetooth, information is fed into the game system that in turn reacts to this and sends back information for different outputs. The different player devices include the Thumin glove (wireless rfid reader), the Stele locator (wireless gps), the Elemental Tablet (tablet pc) and the Urim bracelets (custom wireless sensor/actuator electronics). Several of these are based upon hardware developed within the project. Specifications for those hardware components are only delivered as project internal deliverables but will be made public in the forth coming D7.6.

The part of the game system that takes care of game rules is run on a central game server and controls the overall behaviour of the whole system. This deliverable includes a description of this software as well as descriptions of other server processes providing game state access via a servlet interface, logging to database, and more.
2 THE MOMENTUM GAME

This chapter gives a very brief introduction to the Momentum game. A full game design description will is provided in the D11.7 deliverable. The public deliverable D11.8 will include a concise summary of the game design.

Momentum was a pervasive game about conformity and revolution building upon foundations of larp, MMORPG, urban exploration and alternate reality gaming. The aim was to explore the borderlands between real and ludic, exploring the design space where reality and fiction merge in a seamless, immersive and coherent role-playing experience and test the concepts of the PIMP platform.

The game started in October 2006, when thirty dead radicals possessed a group of gamers. Brought back to life by techno-cult devices, they went on with their struggle to create a better world. The mundane reality was used as the stage for the collective story about personal influence, political change and constructed nature of reality. The game was played continuously for five weeks with varying intensity, mainly taking place in downtown Stockholm, Sweden.

Not all parts of the game and the gameplay are supported by technology. A major game element that was not supported by technology was the different quests within the game. Technology was used to support the struggle for power which is achieved through control of certain regions and by eliminating opposing players. Players will use a device called Stele to locate Nodes that are locations with strong paranormal energy. A Thumin glove is used to locate the exact location of the energy source. By using an Elemental Tablet a group of players may try to take control of a Node. During the first half of the game, players will struggle to find nodes and capture as many as possible. Half way through the game, the gameplay changes and now the task is to capture nodes to build up lines from specific portals in the suburbs of the city into the central headquarters. The winning team of the game will be the group that has created most lines into the headquarters. By using the Urim bracelet players can duel, the losing player will not be able to aid in node capture any more. Thus a team can decrease the powers of an opposing team.

The game artefacts section describes in more detail the role of each technical game artefact. The gameplay overview section describes how the game and the gameplay evolve during the four weeks.

2.1 Gameplay overview

The game starts of slowly around October 1st with a week of introduction. During this week no game play is expected.

October 6-8. The real game starts Friday 6 Oct. During this weekend players will be invited to the clock tower which is their main headquarters during the game. The first scenarios will be introduced. Early morning Sunday 8th Oct the first technology will be used. This will involve the a search for a node with a Stele (GPS). Also a Thumin glove (RFID) and an Elemental tablet (Tablet PC) will be used.

October 9-20. Two-weeks of free-play. During this time the players will continue to capture nodes and mix this with minor quests. At the end of these two weeks it has become obvious that things are not right on the other side. It's getting more and more urgent. There are indications of a major crisis on the otherside.
October 21-22. The third weekend will involve a lot of offensive game play. An external group has organized a traditional LARP that blends into Momentum. One group of players will play this LARP in the town of Katrineholm located 140 km outside of Stockholm. Another group will tear down antennas (our own) in the city, third group has to look for a missing person and the fourth group has to search for a mole with the aid of a hidden map. At the end of the weekend the duelling system is introduced and the bracelets come into play.

October 23-November 3. Second phase of free-play. They can duel and destroy nodes, create energy lines. The strategic level of node capture comes in here – creates an energy net. The tempo of the game will be increased. After one week a person contacts them, he is arranging a party (his dead girlfriend calls him and asks him to arrange a party).

November 4-5. Final weekend, will include a parade for the dead (demonstration) (All Saints Eve), preparations for the party (Friday and Saturday) and the party (Saturday eve) – one room for each group of players. The goal is to get as many people as possible to your room. The end game starts which is about to capture the last node: the element that is on top will be able to get the last node. A final node ritual, a bit special (confining to the standard, but larger and longer).

2.2 Technological Realization

The game system was based upon many different components and processes running on both computers and mobile devices. The central machine was the game server which hosted all game mechanics and also kept the current state of both players and the different astral nodes. Each mobile phone client connected to this server and reported what devices were found around it, such as the tag reading gloves, the GPS units and the tablet PCs. The phone client also relayed information from these into the server that reacted accordingly to the game rules and gave players different kinds of feedback.

The game system was built upon the PIMP platform that is a distributed software platform supporting rapid development of pervasive services, such as city-wide games. Using PIMP, services are divided into functional components and each component is connected through the system to other components. The platform provides means to robustly maintain these connections. The services can use a wide range of devices including sensors and actuators to provide users with a non-screen based experiences.
The core technology for the game was based around different Bluetooth based devices communicating with the central game system through mobile phones. The peripheral devices include a RFID tag reader to discover hidden tags, a GPS to locate specific places and a wearable PC to cast spells with. Furthermore the phone was used to play a lot of different sounds to the players. The technology that should have been the core of a player bracelet was never finished in time. This bracelet would have been the device through which the system would have measured the physical activity of the players and also give them feedback over several more media rather than just sound.

When setting up the system, another process gathered game information from the game master system and fed this into the game server and thus set up the coming game session.

3 IMPORTANT GAME ARTEFACTS

Within the Momentum game there were a number of important game artefacts. This chapter describes briefly each of these in a similar manner. Each section starts to describe the in-game functionality followed by an explanation of how the device is technically and practically realized. Each section ends with a description of how the device is modelled into the game system.

To better understand how the different devices fit into the game structure the following brief explanation is given. All players in the Momentum game are assigned to a specific element, air, water, fire or earth. Each element group within the game has its own Elemental tablet which is used to call upon the powers from their element. There is a total of 4 Elemental tablets within the game. The tablets enable the players to invoke different spells and protections. Another artefact is the Thumin which is a sensor for paranormal energy and is important for locating important sites. Each elemental group has access to two Thumins each, thus a total of 8 in the game. Each element also possesses a Stele, an antenna to focus and amplify paranormal energy. Finally there are 4 Urim bracelets, the primary link between the spirit and the body.
Each player will be given a mobile phone though it does not have a specific role within the game but is required to establish communication between artefacts and the central game server.

3.1 Urim Bracelet - a link to the spirit realm

The in-game description of the Urim explains the Urim bracelet as the primary link between flesh and spirit. Furthermore the game description claims that the Urim measures occult energy (chakra activity, mesmer-fields etc.) of the host. This enchanted object is the house of the possessing soul, a crystalline processor and storage-facility from where the ghost controls the host body. The soul’s influence over the host is amplified through a number of “emitters” affecting the nervous system of the host, ultimately creating a second consciousness co-existing with or overriding the original one. The system has a psychic override switch, allowing the host to stop possession at any point. This compliments the standard procedure where the host is always the final arbiter concerning actions to be taken by the body. The Urim’s programming and design is based on traditional methods of invocation, including the ancient Hebrew device bearing the same name. The Urim comes along with the Urim breastplate, a device for measuring the energy of the heart chakra and the subtle power of breath. The breastplate is worn like a harness directly on the skin of the host’s upper body.

![Urim bracelet.](image)

The Urim bracelets are realized on a hardware platform developed by SICS called uSwarm. In short the uSwarm is a low-power embedded system with a Bluetooth radio for communication. Furthermore it has a wide variety of options for connecting sensors and actuators. The main CPU (NXP LPC2138) on the actual circuit board runs at speeds up to 60 Mhz and has 32K bytes of RAM and 512K bytes of Flash ROM. The Bluetooth chip (Ezurio BISM) provides for communication up to 80 m in direct line of sight with for example a mobile phone. Software on the phone makes the sensors and actuators part of the Momentum game system. The uSwarm is described in the project internal deliverable D7.5 but will be made publicly available in D7.6
For Momentum the uSwarm module is extended with two external boards providing the functionality required by the Urim design. The board called the Bracelet board interfaces and connects the different sensors and actuators. Some sensors and actuators can not be mounted on the actual circuit board since they require ambient stimuli, e.g. the light sensor needs incoming light. These have been fitted on the smaller LED/Sensor board and connected via a wire to the Bracelet board. The different sensors connected to the Bracelet board include:

- Temperature sensor (wire)
- Accelerometer (Bracelet board)
- Breathing sensor (wire, Urim breastplate)
- Microphone/amplitude detection (wire)
- UV sensor (LEDSensor board)
- RGB light sensor (LEDSensor board)
- White light sensor (LEDSensor board)

It is not possible to have both the white light sensor mounted at the same time as the RGB light sensor since they share the same mounting holes. With the use of three independent light sensors, one red, one green and one blue, the RGB sensor gives the possibility to measure different hues of incoming light. This comes at the expense of cost as each red, green and blue sensor cost as much as one white light sensor. Thus one has to decide to use one white light sensor or the three times more expensive RGB sensor but get the extended functionality of measuring hues of ambient light. For Momentum we decided for a simpler solution and thus mounted only a white light sensor.

Furthermore the Bracelet board provides control of the following actuators:

- Three individually controlled vibrators (wire)
- Electric shock generator (wire)
- RGB LED (LEDSensor board)

The electric shock generator is taken from a prankster toy and connected to a software controlled switch on the board and thus controllable by the game mechanics. It generates short electric pulses with a peak to peak voltage about 100V and a current of a few milliampere.
The RGB LED is made of three individually controlled LEDs, one red, one green, and one blue. Each light is controllable to 15 different intensity levels thus the LED can produce over 3000 different colours of varying hue and intensity.

The phone software provides proximity detection of the uSwarm. Once in proximity the phone connects to the uSwarm and starts receiving sensor data. A PimpObject reflects the changes in sensors through a number of properties as well as several properties for the game system to control the actuators.

To aid players in building a model of the functionality of the Urim without explaining the exact technical details the functionality of sensors and actuators are explained in a game consistent way. The vibrators provide tactile feedback that cannot be observed by others, indicates changes in the magical field, giving notice to incoming spells, cast spells, complex effects, which the player must roleplay their response to. The in-game explanation is that you feel the inhabiting soul’s reactions to the changes in the magical field. The electroshocks indicate power drain, the length of the shock signals amount of power lost, the intensity indicates amount of power left. Several short bursts in series indicate a curse has been sustained. From a game perspective it is the inhabiting soul in the bracelet that loses energy. Through the possession link to the player, energy will be taken from the player and that is what he or she feels. The less energy you have the more you feel the effects. The colour of the crystal LED indicates different states of the residing spirit. The breathing sensor which is called the Urim breastplate is explained to be worn in a way that “it should be positioned so that the strap is always tight around the chest, expanding with every breath”. It is explained as being a link between the wearer’s soul through the heart and the breathing.

Mounting of the electrodes for the electric shock turned out to be a major issue. The skin on the arm has much higher resistivity than the skin on the inside of fingers which the prankster toy was designed for. Thus the electric shock was barely felt when wearing the bracelet. The Urim was mainly intended for the player versus player game play and because of the issue with the shocks it could not be used as intended. It was decided to cancel it and emphasize on the player versus environment game play instead. The Urim bracelet has been re-used in the demo game that was developed for the IST conference.

3.2 Stele of Unity – the locator

The Stele is described to the players as a skyward amplifier, an antenna to be raised to the sky whenever possible to channel the powers of a rite towards the stars, hijacking other radio-emitting devices in the vicinity to amplify the signal. It is used to channel energy to specific places.
Technically the Stele is a Bluetooth GPS and is used by the system to detect that players are at a specific physical place to perform some ritual. The software in the phone detects the proximity of the GPS which triggers the phone software to connect to it. The available positioning information is communicated to the central game server through means of connected links. The GPS is represented as a PimpObject in the phone with properties for latitude, longitude and time stamp. These properties are connected to a games rule object in the game server.

The game mechanics monitor the position information received from the four Steles in play and if it enters a region of a node, all players in vicinity of that Stele will hear a sound through their mobile phones. Vicinity to the Stele is judged by the system from which players report proximity to that Stele through the Bluetooth scanning process. This indicates the presence of paranormal energy to the players. An audio PimpObject in the phone is used to play the sounds. Which sound to play is decided by the game server that reacts to the information received from the different GPSs. By updating a property that is connected to the audio PimpObject in the player’s phone the game server decides which sound should be heard for that node.

3.3 Elemental Tablet - The Input device

The in-game functionality of the elemental tablet is to control the powers of paranormal energy. It displays parts of a system designed to channel the paranormal powers of a dead soul into a system. This system is capable of manipulating the invisible realms, ultimately allowing the user to assume command over any part of a creation through control of angelic hierarchies. These celestial beings may be viewed as quantum phenomenon, animistic nature spirits, elementals or divine messengers, their function is still the same. By tracing the hierarchies on the tablet and verbally invoking names, reborn spirits assume command over the invisible world.
Technically the tablet is realized as a Java application on a Tablet PC with Bluetooth. From a system perspective this device is not much different from any other Bluetooth device. The phone detects its proximity and makes a serial connection to it, sending and receiving commands. The players use a pen to trace names of celestial beings on a matrix with 12 rows and columns. The ritual is divided into different steps, an initialization, a calling and a finish. Each of these steps includes a minimum of three separate traces where each trace is at least three characters long. For the ritual to be successful all of these has to be correct! This was not only a demanding task for the players but also the developers while debugging the system. First one has to know and understand the structure of the tablet and then successfully do a ritual which usually took a minute or two to perform.

In the phone the Elemental tablet was represented as a PimpObject with properties that indicated which steps of a ritual had been succeeded. This object also had a number of properties to indicate failures and problems, which were specific commands sent to the Tablet PC. The Java application could for instance lock out the player if a ritual failed, which was decided either by the game masters or the game server.

### 3.4 Thumin – the identifier

Much of the game play, specifically the PvE is about detecting and finding the hidden world of the dead. The Stele provides means for rough area detection for this task but the Thumin will provide most interaction when searching for the hidden world. The in-game description explains the Thumin as seeing connections and lay lines. It is the “dowsing rod” of the game. When the soul dwells in the Urim, its perceptions are limited to those available to the host and the close field monitored by the Urim bracelet. The Thumin-glove extends these senses to include perception of the invisible landscape. Lay lines, nodes, spirits and ghosts can be discovered using the sensory capabilities of the Thumin. The glove works by amplifying psychometric abilities of the ghost and has a very limited range.
The Thumin is realized with a Bluetooth enabled RFID tag reader. A simple embedded microcontroller (BasicX BX24) communicates with a RFID reader chip (IB technology Micro RWD MF). An antenna (7 cm in diameter) is connected to the reader and placed in the palm of the glove. The microcontroller continuously polls the reader for any tags present in the range of the antenna. It keeps track of those present and through the Bluetooth (Ezurio BISM) chip it communicates these changes. The microcontroller also provides feedback to the glove wearer. A vibrator in the glove vibrates in a heartbeat fashion when a tag is present in the field of the antenna. The vibrator is also used to indicate low battery capacity.

By wearing the glove a player can use it to slowly move his or her hand over different objects to detect the inhabited soul powers of the object. The player knows that the Thumin-glove may be limited to extremely short range (5 cm) and that angle to the read object may be an issue. As other Bluetooth devices, the phone software connects to it when in proximity and reflects information of it into a PimpObject. This object has one property that indicates the last read tag id number and is connected to an object in the game server.

4  **CORE MOMENTUM SOFTWARE**

This section discusses the processes which make out the machinery of the ubicomp reactive technology in Momentum, which is what we refer to as the core platform. Other external hardware and software used to create the richness of interaction in Momentum can be found on the post-mortem site ([http://prosopopeia.se/](http://prosopopeia.se/)) and in the upcoming evaluation deliverable.

Much of the basic distributed software issues, such as for example game state synchronization, relies on the underlying PART platform, which can be studied in more detail in deliverable D6.7. The following diagram gives an overview of the core Momentum software components.

Briefly, the **Session Server** process works as an initial point of contact for all other pimp processes (though PART does not require this). The **Game Server** handles game event logic and communicates mostly with the phone clients which provide input and output from players. The **Starter** process initializes the game state, as a fallback to being initialized by a web application via the **Servlet**. The **Servlet** offers an http request interface to interact with the PIMP system, whereas the SQL Logger offers a classic database view of the current and log of game state.
4.1 Phone Client

As explained in the Technological Realization section, all the active game artefacts described in Important Game Artefacts communicate with the J2ME phone client software running on the phone via bluetooth. The most important features of the phone client as it was used in Momentum were

- game state communication with the game server
- bluetooth device communication and data processing
- audio output

To recap, the phone client continuously “scans”\(^1\) for bluetooth devices at a configurable interval. When discovered, the phone client automatically connects to (pre-configured) the devices and supplies data to the pimp objects resident in the pimp process, which then processes this data and updates the shared game state. The phone client in the Momentum game communicates with the

- Urim sensor/actuator bracelet
- Stele gps locator
- Elemental tablet
- Thumin rfid glove

Furthermore, it scans for presence of the other players' phones.

\(^1\)Not a traditional bluetooth device inquiry scan. See The use of bluetooth section.
All these devices have corresponding pimp objects in the phone client with Java class code\(^2\) which processed the data from the devices and updated their shared properties (see e.g. The GPS Object in the figure in Appendix A). As explained in earlier sections, some of the devices also had output capability, e.g. the Thumin glove had a vibration component. Furthermore, the most important user interaction mechanism was the audio feedback using the phone headset. The phone client had an audio object, capable of playing back both locally stored audio files as well as downloadable files.

The property data was propagated to the game server using the PART subscription and persistent tcp connection concepts. Configuration of the phone client was done through the use of a one-time login mechanism using a GUI on the phone screen and subsequently downloaded configuration files (see Setting up and running ... section).

4.2 Game Server

The responsibility of the game server is to administer the rules of the Momentum game and their effect on game objects. The server is implemented as a Java SE PIMP process that is connected via the session server (see Session server section) to all active mobile phones.

4.2.1 Objects used by the game server

To be able to reason about the game's state and rules the server is populated with necessary objects. Most of the objects are created in the server by the starter process (see Starter section) either at startup, e.g. game content, or when players connect for the first time, e.g. player information and stats. The objects are modelled as PIMP objects containing properties for all pieces of information. To see the properties used by objects see appendix A. Following is a description of the different object types residing on the game server during a game.

- **Node Objects**
  A part of the Momentum game is built around control of regions in the game. This is done by finding a node using a gps and rfid reader and to take control of it using the tablet PC. Hence information such as position in the form of latitude/longitude, associated rfid-tags and current owner are important properties of a node. During the Momentum game there were 50 node objects on the server, although not all of them were used during the game. The extra ones were there to allow for quick addition of new nodes. Those were "disabled" by having their position set to some unattainable value.

- **Shadow Player Objects**
  The information and capabilities of a player, in the form of several objects residing on the player's mobile phone, is aggregated in the game server in the Shadow Player Object. An important part of the Shadow Player Object is to convey to the server if the player is in proximity of any of the games bluetooth device artefacts. The server also uses this object to issue the phone to play audio clips.

- **Multiplex Objects**
  Every player in a group or faction may potentially be a carrier of devices like the gps or rfid-glove that belong to the group. In addition, players may decide to handover

---

\(^2\)There was also an option of using dynamically updatable Hecl scripts ([http://www.hecl.org/](http://www.hecl.org/)) which was never used due to time constraints.
devices now and then between the members of their group. However, from the point of view of the game server, the current carrier of a device is of no importance to how the rules should be applied. To be able to remove the burden of device owner administration, and link creation and destruction in the PIMP model, multiplex objects were introduced in PIMP. A multiplex object is a sort of transform object that joins several properties into one. In this way all players of a group have their device properties linked to that device's multiplex object in the game server and it doesn't matter which player's phone that happens to be in touch with a device, the update will propagate to the output of the multiplex object. These multiplex objects can grow dynamically which is a feature used by the starter process when it is setting up links for a new player (see Starter section - "a phone connects"). In a running Momentum game there are one gps mux, two rfid-glove muxes and a tablet mux for each faction totalling 16 multiplex objects.

- **Ritual Objects**
  The game server contains a ritual object for each of the four factions in the game. These objects track the state of rituals, involved players and the outcome. The ritual object also administer a timer after successful ritual during which the game server waits for game master intervention before issuing the effects of a completed ritual to the rest of the system.

- **Audio Store**
  A lot of the feedback to the players of the Momentum game came in the form of audio clips played on the mobile phones. The command to play a clip is issued from the game server in the form of an URI string. To be able to modify these URIs on the fly through the normal PIMP mechanism (servlet API, links, browser) they were included in a game server audio object as pimp source properties. The base of this URI, e.g. "http://www.sics.se/momentum" or "file:///c:/Other/Momentum/", was separated as a separate property to allow for quick relocation of the audio clip repository. In the Momentum game the audio files were stored on the phones. The audio store object contains URIs for clips such as node state sounds, ritual progression sounds and group/faction themes.

- **SMS Notification**
  The game server used PIMPs sms object to send notifications to game masters at important steps in the games progress, e.g. a faction starting a ritual. The list of phone numbers used in the notification service was stored in a separate object, aptly called Phone Number Store, to allow quick management of sms receivers.

### 4.2.2 Game Rules

The role of the game server is to administer the rules of the Momentum game. Some mechanisms of the game like node searching and location using gps and rfid-glove may be resolved in a single step with a fairly limited number of participating objects. Other mechanisms like the multiple stages of a node ritual is broken down into a number of smaller rules each responsible for a part of the evaluation and progression needed to be performed during node rituals. The rule engine is implemented using a dispatcher that is triggered when one of the game affecting properties changes its value. The dispatcher invokes the appropriate code responsible for processing a rule. In the Momentum game most rules are triggered from new values originating from one of the game artefacts or from changes reported by players bluetooth proximity detection machinery running on their mobile phones.
There are three parts included in the administration of most game rules in the server:

1. **Find the appropriate objects.**
   The first part when processing a rule is a filtering mechanism that finds the appropriate object or objects needed. This may for example be iterating through all node objects to find a node with an rfid-tag matching what a glove is reading or finding all players in proximity of the earth tablet PC. This filtering is done through PIMP property name-value matching, e.g. give me all objects of type 'Shadow Player Object' that has its property 'Air Tablet Near' set to true.

2. **State reasoning**
   In most cases this is a step by step test of the participating objects states against the rule being processed. For example checking that a player is close to a tablet being used in a ritual when the tablet magus just traced the participants name on the tablet.

3. **Conveying the outcome**
   The result of a processed rule is propagated back into the game system by setting properties in objects. This may in turn cause the game server to start processing rules that are triggered by the new state of the game. A lot of the rules in the game uses the 'Audio' property in the Shadow Player Object to issue feedback sounds to the players.

### 4.2.3 Game Master Intervention

Based on the distributed PART/PIMP platform, the Momentum game server is by default open to remote on-the-fly adjustment of the values of properties in game objects. This is a feature that may be used by game masters to control or adjust the state of the game. A node may for example be repositioned or an occupied node may become a free target by clearing its 'Owner' property. In addition, GM intervention has been incorporated as a part of the end of a successful ritual. When a ritual starts the GMs get an SMS notification giving them a heads up to the possibility of controlling the ritual outcome. If the group performing the ritual proceeds according to the rules of a ritual, i.e. carrying out a technically correct ritual, the game server waits for a GM-configurable time before triggering the success outcome. During this time the GMs may inspect the log of the ritual and decide if it should succeed or fail. If a GM decides to intervene in either way he uses the PIMP servlet API or the PART browser to set a special property in the groups ritual object. This causes the game server to trigger the last part of the ritual issuing either a succeeded or failed ritual.

### 4.3 Starter

When the game server is started it has no knowledge of mobile phones, players or any sort of connections to them. The role of the starter application is to react when the game server or the mobile phones connect to the session server for the first time. When this happens it populates the game server with appropriate objects for the game and creates PIMP links between the game server and each mobile phone. In the game server it creates gps, rfid-glove, and tablet muxes for each faction as well as the ritual, shadow player, and node objects.
5 DISCUSSION

During development of the core platform for the Momentum game a number of design choices were made and later on during gaming several lessons learned. In this section we will discuss some of them and also touch upon some directions for future development.

5.1 Reconfigurability

One of the positive findings in the pre-game content adjustment phase was the advantage of having the means to dynamically change game content such as node names and latitude/longitude. This was done on several occasions with test players roaming the game area testing node discovery and communicating node data errors to someone that had access to a browser were the PIMP servlet was used to alter the node data. If the servlet was to be extended to supply a mobile phone friendly interface the test players themselves could adjust the game data making for a more efficient test phase.

5.2 Hard coded game rules

The game server in the Momentum game implemented the game rules as a centralised set of custom designed Java code (see Game Rules section). A drawback of this hard coded approach is that if a need to change a rule arises, or if a bug is found, the server must be stopped and reprogrammed. An interesting alternative to that approach would be to express the rules using the mechanisms supplied by PIMP, i.e. transform links and embedded scripts. This would enable a more decentralised server and possibly load-balancing if needed and, more importantly, an on the fly reconfigurable game server. However, when inspecting the three phases of a game rule implementation as described above (see Game Rules section) and looking at the mechanisms supplied by PIMP there is a lack of support for the first filtering phase. A potential way to have reconfigurable game rules is to implement each rule as one or more embedded scripts and have some sort of distributed pattern matching for finding the objects needed to perform game rule reasoning. Even though this was the initial design, the lack of time during the Momentum core platform implementation ruled out this design in favour of the more straightforward hard coded design.

5.3 Debugging tools

One obvious and foreseen lesson concerned the need for allocating considerable resources for creating proper debugging/analysis/performance tools. However, being very short on resources, it was one of the activities which were cut short.

The development that did happen, for instance the implementation of a phone client pimp object simulator, did speed up certain test phases. Also, as mentioned in Reconfigurability above, being able to use a servlet interface for poking around inside the running pimp system proved very useful.

5.4 The use of Bluetooth

Bluetooth is developed to set up a connection to a specific device and keep that connection while the device is in use and then disconnect. The model in Momentum is
very different. The game design required both dynamic connections to all different devices. Some devices were personal while others belonged to a group or anyone could use it. Also the game rules relied upon proximity detection of players. In this section we describe how we solved this and the issues we uncovered.

To detect proximity of a Bluetooth enabled device, the Bluetooth specification provides with device discovery mechanisms. Though none of the mobile phones\textsuperscript{3} we have tested on the market supports to keep a connection while performing a device discovery. One can query the Java Bluetooth implementation if this functionality is provided or not by reading the local properties 

\texttt{bluetooth.connected.page.scan} and 

\texttt{bluetooth.connected.inquiry.scan}, see the JSR-82 documentation for more details. To perform a device discovery takes at least 20 seconds and usually longer, thus if we would have used this it would mean every time the system would scan we could not communicate with any of the player devices; i.e. the players would not be able to play during these moments. 20 seconds is not a very long time but since we have a distributed system and specifically communicated over GPRS and UMTS there is no way to synchronize these scans between all devices. Thus we cannot guarantee that a player action will be monitored by the system. We needed another way of performing device discovery while being able to communicate.

Since all devices within our game are known before the actual game session there is another option which is the one we used. We can try to make a connection to the device and if we are successful we know it is in the proximity. If we do not to further communicate with the device, we close the connection to allow others to perform proximity detection to it. Though there exist a couple of cases where this will not work.

- If a device is already connected to a phone.
- If one phone is already probing a phone or a device for proximity.

A device that initiates connections is a Master. A Master can connect to up to 7 Slaves though a Slave can only have one connection. A Master can have one incoming connection from another Master. Our phones are Masters since they are probing for connections to other phones and other devices. The devices become Slaves. Thus only one phone at a time can be connected to a device and only one phone at a time can probe for proximity on the same other phone.

Fortunately it is possible to tell the difference if a connection attempt failed due to the device was not in proximity or if the connection was not allowed due to the above mentioned circumstances. It is done by measuring the time it will take for the system to respond with the exception. If the device is not near it will take at least 5000 milliseconds. If the probed device is busy an exception will be thrown within a time span of 500 to 2500 milliseconds.

To complicate even further there is the maximum number of active connections a phone can have. This is not necessary 7 as is the maximum defined by the Bluetooth standard but can be much fewer.

Due to the time it will take to scan for devices not in vicinity it was a bad strategy to scan for all the different phones to determine if a group of player where close to each other. In the game rules, proximity is only important while using some of the devices like the Stele and the Elemental Tablet. Thus it is much more time conservative to scan

\textsuperscript{3} SonyEricsson K750 and K800
for only these, which are much fewer than the number of players. Though it might take almost a minute to determine if a player is close to one Stele (3 missing Steles and 4 missing tablets gives a time out of a total of 35 seconds).

Between each connection attempt it was also necessary to provide some time out or other threads in the application on the phone would not have enough time to finish their execution.

5.5 System complexity

The system had not been tested in any large scale real case before and several major bugs were found when setting up the system the first time. Due to several factors it took several days of testing to find each bug. The major factors for the long and slow debugging were:

- the complexity of the whole system
- clients running in devices with very little support for real time debugging
- the physical distribution of the whole system
- the great number of processes, to just start a test case could take 30 minutes.

The design of technology met several challenges on the way. The design process started with gathering requirements from the game design group, to start to understand what they wanted to achieve. Not everything could be included due to limited time and resources and it was a hard struggle to decide what to include. To this end, we invented a point system, which proved as a great way to communicate to the game designers the complexity and uncertainty of different technologies. Every technology was given a certain number of points, the higher the more complex/effort, and then the sum of all technologies the designers wanted to include had to stay under a certain threshold. After gathering an initial system specification for the game the implementation started, including both developing completely new hardware as well as extending an existing software platform and implement the game mechanics. The complexity of the hardware turned out to be a bit too complex. Unfortunately this was among the most interesting parts, specifically from an interaction perspective. A lot of effort was put in to try to solve the problems but at the end, we realized if we wanted to host a game, we needed to change our focus and implement support first for the less complex technology.

5.6 Managing resources

With 30 mobile phones each with individual subscriptions, 4 gps units, 8 gloves and 4 tablet PCs management of these devices and the installed software became a task larger than anticipated. The device management included not only identifying which device was which one but also managing configuration of them, installed software, and who had borrowed it and where it might be physically.

Each phone was given a unique name to be able to both easily identify the correct phone but also to be able to easily talk about that device. The naming was based on the phone’s player element and a sequential number, e.g. Water4. The phone’s Bluetooth identification name was set to this name. The game design required each phone to be personally adapted to each player, i.e. an in-game description was given that each phone had been adapted to the frequencies of each individual. This was achieved by setting an
individual background on the display of each phone which included the player’s in-game name (names of historical people). One problem was that this name was not the same as the name given to the phone, due to the fact that the in-game name was never needed in the technical system.

Each phone had to be configured correctly for GPRS and 3G to get networking up and running. This included registering the phone on the operator’s website and receiving and installing settings via SMS. The configuration and settings section of the operator’s website experienced downtime periods frequently which delayed the process to setup the phones. To setup, activate and configure the phones took almost one full day for one person.

To minimize delays and network costs each phone stored the played sounds locally. This also required transferring all sounds to each phone, which took about 5 minutes each to copy and paste these files over to the phone.

The final major concern regarding resource management was software installation and different versions. Software was installed on phones, tablet PCs, gloves and in the game server. If any of these had a different version this could possibly cause communication problems and thus introduce bugs into the system that nor really existed. To complicate matters even more there were several persons developing and bug fixing different parts of the system at the same time, so sometimes the game server would become a scarce resource since people might be working on different versions of the system.

To keep track of the progress and status of all the equipment an editable web page was set up. Here the group could keep record of device id numbers, device names, possibly the player the device belonged to, software versions and who might have borrowed it.

6 PIMP – Pervasive Interactive Mobile Platform

The Pervasive Interactive Mobile Platform (PIMP) is primarily a toolbox for building applications based around mobile phones with portable external sensor and actuator electronics connected via Bluetooth.

6.1 PIMP concept

PIMP provides an abstraction model to these sensors, actuators, and other system components in the form of inter-connected objects, called PimpObjects. Objects exist in different clients and each provides different or client specific functionality, e.g. each player has a player object in their phone. The second part of the abstraction model is the ability to set up links between these objects. A link is established between two properties of two objects. When the source property changes, the link updates the property value of the drain property for that link. That is the basic principle how information is exchanged between different objects and applications built upon PIMP.

PIMP is built on top of PART (Pervasive Applications RunTime, see the infrastructure workpackage, deliverables D6.3 and D6.7). PIMP utilizes PARt’s protocol-independent persistent data connections and support for creating distributed game object states. Through these mechanisms the PIMP links are created and maintained. Furthermore the distributed object state mechanism is used to communicate information over the links.

The PIMP software currently contains a number of components:
• J2ME midlet
  A fairly generic PIMP peer which handles communication with sensor/actuator hardware. The application can download software and hardware configurations with additional on-device configuration. It can also handle dynamically detecting other known Bluetooth devices, and using Bluetooth GPS and RFID devices.

• J2SE session lookup server
  Initial point of contact for PIMP peers. Also keeps tabs on connected peers and can send this info to interested peers.

• http servlet command software
  Offers a limited PIMP/PART command set via http to perform basic operations such as managing property links, manipulating properties, managing PIMP objects, etc. This is one of a few different ways of external accessing the PIMP system. The current solution uses the Apache Tomcat Java servlet reference implementation.

• J2SE graphical configuration software
  Basic graphical GUI primarily for viewing available objects. It also lets the user create links between properties letting PIMP take care of property value propagation, as well as creating new objects for standard logical operations (e.g. thresholding).

• JDBC logging software
  PIMP process which can log various PIMP object data to a database via JDBC. Together with the servlet interface, this yields the option of using any web-based technologies to access a PIMP system.

• Text logging software
  A simple means to create rudimentary logs from a PIMP system. Simple command line based application that prints out newly connected or disconnected clients, shared object information and changes of properties.

Again, the core of PIMP offers a way to represent real-world sensors and actuators as shared objects, as well as linking properties (typically sensor and actuator values), typically via simpler logical processing objects (e.g. sensor → threshold → actuator).

Linking properties provides a way of making logical data propagation chains between objects. A link between a source and a destination objects' properties can be set up via the servlet http interface or by creating a specific PART network event. This event, containing information on object and property ids, is then sent to the process with the destination object, which then creates a subscription to the source object and makes sure property updates get propagated according to given criteria (e.g. max update frequencies).

One of the preferred PIMP processes, is the PimpPeer MIDlet. This application runs on mobile devices such as mobile phones and PDAs. Through its settings files it allows for simple configuration of the PimpObjects which should be created on the device by default. Furthermore, it handles scanning for, configuring, and communicating with different Bluetooth hardware.
6.2 Discussion

6.2.1 Status

Current version of PIMP is stable enough to run a game for several days that has been implemented with PIMP. Though the current version of the PIMP runtime includes a bit too much, e.g. support classes for scanning for Bluetooth devices and a generic serial communication library. The platform has not yet been used to create any complex setups where links are dynamically changed. Nor has the current version been tested towards any benchmarking requirements.

6.2.2 Strengths

The major strength of PIMP is its model for how services are created by combinations of independent components that can exist anywhere. Correctly designed systems can be opened up towards the end-user who can on a top level start to examine different components that make up a service. If required the user can easily remove or set up new connections between components. The model also supports hierarchical modelling thus an end-user can decide to examine one component in more detail by opening it up and see what sub-components it is made up of and how these are inter-connected.

6.2.3 Weaknesses

The major weakness of the platform is that it requires careful design and a new model to learn for the application developer. A simpler and a more straight forward solution seem more attractive in the beginning but once the different components starts to get re-used the strengths become apparent.

6.2.4 Goals and plans

The immediate plans are to move parts of the current implementation that doesn't specifically implement PIMP functionality out of the pimp core. The plan is to minimize the core needed to implement a PIMP enabled application.

Another issue with the current software structure is that a PimpObject only implemented using Java 2 ME classes can not be accessed in any Java 2 SE implementation since the underlying code is trying to create an instance of the object in the local process when establishing links. In the PIMP model instances of objects reside in specific process and are instantiated in those processes and thus from a PIMP perspective it would make sense to write an object that can only exist in a mobile phone.

Hierarchical modelling of components is not yet supported by the software. A future version should provide means to define a set of interconnected components as a super component as well as define what properties it exports as sources and drains.

Also, the final purpose of the J2ME MIDlet is not set. Currently it's being used as a generic runtime with support for some specific external devices. This should be clarified and a plan for a plugin behaviour should be defined and implemented.
7 OTHER SOFTWARE COMPONENTS

This section covers the role of rest of the components and describes their role in the game system. These include typically backend and debugging resources.

The SMSObject resides on the server and acts as a means for the game system to alert Game Masters about important events, e.g. when a group of players initiates to take control of a node.

To allow game masters to observe and do changes in the game state a orchestration tool is being developed. To provide an easy interface for this web-based orchestration tool two different solutions has been developed. The JDBC logger provides easy access of the current game state as well as a log history. Basically the database provides different tables for objects and property changes. Thus the orchestration tool can query about the current game state. Through the servlet interface the orchestration tool can change and influence the game system by issuing specific PIMP commands such as setting property values and creating links between objects.

The text logger together with the simple graphical editor provides means for debugging parts of the game system, through the possibility to quickly create links between objects and observe changes of properties. The graphical editor allows one to select objects in any running process. For each object a list of drain or source properties are given depending if the object was selected on the Sources side or the Drains side, see Figure 7: The simple graphical editor.

By selecting a source and a drain and then press the Create link! button a link will be created between these two. Transformers Links does not currently work and is thus not described here.

8 ORCHESTRATION TOOLS

The orchestration tool is an information overview and decision making tool for the Momentum game. The tool is targeted at non technical game-masters and the interface is therefore adapted to create an easy and fast work flow for them.

The orchestration tool contains information about the players, the characters, the game setup, the locations and the props used in-game. This data is aggregated with all information flowing through the PIMP network to create sensible events and enable tracking of what the players have done and in what way they are interacting with the game,
Figure 8: Different components used by the orchestration tool.

The orchestration tool is a web-based application written in Perl, connected to a postgres SQL backend and connected to PIMP through another SQL server and the servlet interface. The postgres server stores all non-PIMP based game information, such as player names, real world names, means of contact etc.

9 SETTING UP AND RUNNING THE MOMENTUM GAME SYSTEM

Provided with this deliverable is a zip file including the Momentum game system. The following instructions will guide someone on how to setup and start the Momentum game system components included in this zip-file. **NOTE:** since much of the game play for Momentum is based on specific hardware, media etc it is not our intention that anyone should be able to set up a runnable game using this documentation and accompanying software.

Running the Momentum setup requires you to have the following equipment and software:

- At least one PC running Windows XP SP2
- Tomcat 5.5.17 or similar Java servlet server. Has been tested only on Tomcat 5.5.17
- Java Runtime 1.5 or higher.
- Momentum zip file, linked from the site http://www.sics.se/ice/projects/pimp/

Furthermore some components have specific requirements:

- The SQL logger requires a MySQL server.
- The phone client requires a Java capable phone with JSR82 (Java Bluetooth API) support
- The phone client can also be run on a Pocket PC with Windows Mobile 2005
- A web server accessible either from the mobile phone or the Pocket PC is also required to run the phone client.
- optionally one Bluetooth GPS with NMEA support

The order of which these different components are started is not crucial, for instance you can start the phone client before the session server. Though note when reading this walk through the first time, follow the order given here since some steps require specific
information that is achieved in a previous step. E.g. the address to the machine running the session server is needed when configuring the phone client.

Start with un-zipping the distribution file. It will extract into a main directory which includes a directory called run and one called lib. “run” includes all files needed to start an application, while “lib” includes external libraries such as PIMP and PART.

9.1 Modest server side startup

Presented here is a very simple startup example for some of the server processes. These processes will be explained in more details in sections below. With the supplied momentum zip file unzipped, do the following to startup the session server, game server, starter, and a crude object/property visualiser, just to be able to see some of the objects/properties created at the game server at startup:

In the java/run/ directory use the

1. start_session_server.bat to start the session server
2. start_node_server-nosms.bat to start the game server
3. start_starter.bat to start the starter
4. start_configurator.bat to start a very crude unofficial property linking tool, where you can use the drop-down lists to see objects and their properties. For instance, enlarge the window to see all menus, then choose one of the Node objects in the drop-down below the Drains label. This will then enable you to view all its drain properties in the drop-down below this, e.g. latitude and longitude for its physical location.

9.2 Session server

The session server provides a common communication point for all processes. Since PIMP is specifically designed to run on mobile phones a session server is needed to establish communication since hardly any mobile phone operator allows external connections into a phone. To start the session server locate the start_sessionserver.bat file in the run/ directory and double click it to start the Session Server. This should bring up a window with a text output similar to this:

```
D:\home\kalle\momentum\run>java -cp ..\pimp_j2se.jar;..\lib\mogile_j2se.jar;..\lib\part_j2se.jar org.iperg.pimp.session.SessionServer
SessionServer: Received an event of type PropAdd
SessionServer: Received an event of type PropAdd
SessionServer: Received an event of type PropAdd
SessionServer: Received an event of type PropAdd
SessionServer: Received an event of type PropUpdate
SessionServer: Received an event of type PropUpdate
SessionServer: Listening to address tcp://localhost:1000
SessionServer: Received an event of type PropUpdate
```

What is important to note from this step is the IP-address or internet address of your machine. This is needed for all your other clients since they will connect to the session server. If you are using a mobile phone the machine needs to be public accessible over the internet. If you are using a firewall you need to allow incoming TCP connections on port 1000. If you want to change the port used, see next section, Simple text logger, about how to configure network settings. If you are running the Session server on a non-
public network you will not be able to use the phone client. The PocketPC client will work as long as the Pocket PC uses a wireless LAN that is on the same network as the machine you started the Session Server on.

9.3 Game Server

9.3.1 Command line arguments
The game server process only has one optional '-clear' argument, in order to specify that the persisted game state should be cleared. Thus, starting the game server is done by the following:

```java
java -cp momentum_j2se.jar;pimp_j2se.jar;part_j2se.jar;rms-20060908_j2se.jar -ea org.iperg.momentum.peer.NodeServer -clear
```

9.4 Starter

9.4.1 Command line arguments
The starter process has the following command line arguments: '-nodeserver', '-procfilter', and '-propfile'. If an id to a running game server is supplied with the nodeserver argument the node data will not be updated. The procfilter argument accepts regular expressions on process names as a way to point out which processes should be handled by the starter.

Example:

```java
java -cp momentum_j2se.jar;pimp_j2se.jar;part_j2se.jar -ea org.iperg.momentum.peer.Starter -nodeserver <node server id>
-procfilter Fire1_Phone
```

9.4.2 Starter property file
To allow easy management of the initial values of these objects, a property file may be supplied as a command line argument when starting the starter process. This propfile is formatted according to standard Java property file syntax with names corresponding to the objects PIMP property names.

This is the standard way of setting for instance the latitude and longitude properties of a node in the game or the magical names of a player in the shadow player object residing in the game server.

9.4.3 A phone connects
When a mobile phone connects for the first time the starter creates links from the Player object on the phone to the shadow player object on the game server. It also connects this phones gps, rfid, and tablet objects properties to entries in the appropriate faction muxes on the game server.

The starter also keep track of potential problems (e.g. link creation timeout) during the setup phase.

9.4.4 Start sequence
The order in which the games all processes is to start is the following:
1. session server
2. starter
3. game server, a.k.a node server
4. mobile phones

9.5 Phone client

The phone client is the main client for including sensors and actuators into a PIMP setup. There exists a Windows client that could provide similar functionality but this is not usually the case specifically for this game where all devices are mobile and are expected to be used throughout the city. Without actual hardware there is very limited functionality provided by the phone client, though some simple tests can be performed with a Bluetooth GPS.

For this setup you will require a mobile phone with JSR-82 support (Java Bluetooth API) and web server to store some simple configuration files.

1. Copy configuration files. In the run/ directory there are three files you need to copy to your webserver, default.settings, default.mogiles, mogiles.config. It is not crucial where you put them as long as they are accessible from any public client. The phone will download these when started. Remember where you put the files, the URL to this location is needed in the next step. The two Mogile files are not really needed for this game setup since no Mogiles are used in the game. Mogile is a simple hardware module for simple sensors and actuators. More information is available here http://www.sics.se/ice/projects/mogile/

2. Edit configuration. You need to edit the settings file to fit your setup. You can do that on the phone client but usually this is much easier from a PC. The following fields are eligible in the settings file:

   - processName – the name of the process running in the phone.
   - initialConnectionUrl – URL to the session server, tcp://your.machine.org:1000
   - useSavedState – yes or no, if yes, the application will persist the state of all objects, thus if the application crashes or you need to turn off the phone, once the application is restarted all objects will regain their previous state.
   - loadObjects – list of classes of object to load and initialize at start-up.
   - mogileMapping – Mogile mapping file to use, should be the name of the file you copied to your web server originally called default.mogiles.
   - mogileConfig – mogile configuration file to use, should be the name of the file you copied to your web server originally called mogiles.config.
   - tabletBluetoothURLs – comma separated list or URLs to tablets. A TabletObject will be created for each.
   - proximityBluetoothURLs – comma separated list of URLs to Bluetooth devices supporting the SerialPortProfile, e.g. a mobile phone or a GPS. A ProximityObject will be created for each.
• gpsBluetoothURLs – comma separated list of URLs to Bluetooth GPS’s. A GPSObject will be created for each.

• braceletBluetoothURLs – comma separated list of URLs to Urim bracelets. A BraceletObject will be created for each.

• rfidBluetoothURLs – comma separated list of URLs to Thumin RFID gloves. A RFIDObject will be created for each.

All Bluetooth URLs have the same format; btspp://XXXXXXXXXXXXX:1 where XXX… is the Bluetooth address of the device. In Windows bring up your Bluetooth explorer and let it search for devices in range. For a found device, right click and select Properties. In the new window you should see something like Device Address followed by a list of number separated by colons. This is the Bluetooth address, add that but leave out the colons.

3. Install the application on the phone. A Java application on a phone is called a midlet. For the application to work the phone needs to have support for Bluetooth in Java. This is provided through the API called JSR-82, check the web site for the manufacturer of your phone if this is supported. The software has been tested on SonyEricsson K750 and K800 successfully and should work on other phones supporting JSR-82. To install the midlet, copy the momentum_j2me.jar and momentum_j2me.jad to your phone and select install.

4. Download settings. The first time you start the Momentum midlet there will not exist any stored setting and the user will be informed that no settings were found through a warning message. Press OK to get to the next screen where you enter the settings file to download.

At the Settings Download screen enter client name. The client name is the same as the filename of the settings file you uploaded to your web server, without the “.settings” file extension. Most likely the client name will be “default” if you did not change the filename. Change Base URL to the URL of the directory where you store the settings file and the mogile files. Press Get Clnt Se menu option to download the settings file. Once downloaded the midlet will show the contents of the file. If there is anything you want to change you can do that now by selecting the appropriate field and change it.

Once all the steps of the setup have been performed, Momentum will use these settings each time it starts. To reload a new setting, enter the key sequence 3-5-2-7-7 (elarp) on the mobile phone keyboard while the splash screen is showing and this will take you to the initial settings screen.

5. Press Get Mog C menu option to download the Mogile configuration file. It will be displayed once downloaded.

6. Press Get Mog menu option to download the Mogile mapping file, once downloaded it will be displayed.

7. Optionally scan for Mogiles. Now you are ready to scan for your Mogiles. If you do not possess any Mogiles it is not necessary to perform this step, just select Go to skip the scanning. Select Scan in the menu to start scanning. This may take some time. If you do not get any devices listed, you can scan one more time. If no Mogiles were found make sure you have Bluetooth turned on on your device. After selecting Mogiles to use, press Go.
8. Connect to the SessionServer. Next you should see the PIMP Visualiser. If you have selected any Mogiles you should now see a simple graphical user interface to present either data from the Mogile or to control actuators. If you did not select any Mogiles or skipped the scanning step you will be presented with a blank screen. In either case there should be a Connect option and Exit. Press Connect to connect to your session server. In the session server you should see the phone connecting.

9.6 SQL logger

With the SQL logger it is possible to log events from the game system in a SQL database which can be used by different applications or services. In our game system the orchestration tool uses the information to present the current game state to the game masters. To run the SQL logger you need of course a SQL database and also a matching JDBC implementation. We have used MySQL and their JDBC implementation.

1. Download the JDBC jar (http://dev.mysql.com/downloads/connector/j/3.1.html) and place it in the lib/ directory. The script to start the SQLLogger, SQLLogger.bat in the run/ directory expects this jar to be called “mysql-connector-java-3.1.12-bin.jar”. If you downloaded another version, open the script and change it to fit your file.

2. Set up a database. Before you can start the SQLLogger you need to setup a database for it and a user with access rights to INSERT. To get the correct structure there is a SQL script provided that creates the correct tables in a database. The script is called log_db_default.sql. Create a database and then select it and run this script on it to create all needed tables. You can load this script into MySQL with some administration tool like phpMyAdmin.

3. Configure the SQLLogger. There is a properties file in the run/ directory for this, called sql_properties.xml. It looks something like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>SQL server settings for SQLLogger.</comment>
  <entry key="ServerHost">localhost</entry>
  <entry key="ServerPort">3306</entry>
  <entry key="Database">pimp_testlog</entry>
  <entry key="DatabaseUser">pimp_logger</entry>
  <entry key="UserPassword">momentum</entry>
</properties>
```

Change values to fit you setup.

4. Start the SQLLogger by executing the SQLLogger.bat script in the run/ directory. You will get an output similar to:

```
D:\home\kalle\coding\iperg\elarp\pimp\java\run>java -cp ..\momentum_j2se.jar;..\lib\pimp_j2se.jar;..\lib\mogile_j2se.jar ..\lib\part_j2se.jar lib/mysql-connector-java-3.1.12-bin.jar org.iperg.pimp.utils.SQLLogger
SQL Logger starting.
Initializing JDBC connection...
Register as a logger
PimpLoggerManager: Connecting to server using url
```
Look at the output window of the Session Server and you will note another configurator process has connected. PIMP only makes a distinction of two different kinds of processes, either clients or configurators. A client is not informed about other connecting clients. A configurator is continuously informed about newly connected processes, both clients and configurators. This is taken care of by the SessionServer.

SessionServer: Connection established from 1158677795312:0:1:0
SessionServer: New configurator connecting 1158677795312:0:1:0

9.7 Servlet interface

The servlet interface allows external application to communicate with PimpObjects through a simple URL based interface. Applications can set or read properties, create or remove objects, and create links between objects.

9.7.1 Requirements

You need to have Apache Tomcat installed to run the servlet. Apache Tomcat is a servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. Go to http://tomcat.apache.org/ and check requirements, download, and install (At the time of this writing, we're using ver 5.5.17.)

9.7.2 Setup

Once Tomcat is installed the specific PimpServlet files need to be installed in the Tomcat 'webapps' directory. You need to create a PIMP servlet directory tree in Tomcat's 'webapps' directory. These directories are needed to be created (assuming TOMCAT_HOME/webapps already exists):

TOMCAT_HOME/webapps/elarp
TOMCAT_HOME/webapps/elarp/WEB-INF
TOMCAT_HOME/webapps/elarp/lib

Copy the 'momentum_j2se.jar', 'lib/pimp_serlvet.jar', 'lib/part_j2se.jar' files to the 'TOMCAT_HOME/webapps/elarp/lib' directory. These jar files contain all classes needed to run the PIMP servlet. The 'web.xml' file should be copied to the 'TOMCAT_HOME/webapps/elarp/WEB-INF' directory.

9.7.3 Starting

On Windows there are two options for starting Tomcat depending which version you downloaded. If you chose the Windows Executable Tomcat will be installed as a service. In the taskbar you will have a Tomcat icon. If Tomcat is already running, chose Stop service. To start Tomcat select Start service.

If you downloaded any of the archives instead you need to use a start script instead. Start the Tomcat with the command file '<tomcat>/bin/startup.bat'. The PIMP servlet classes are loaded automatically by Tomcat when it starts. Similarly, stop Tomcat (and the servlet) via '<tomcat>/bin/shutdown.bat'.
Now the PIMP servlet should be up and running. The easiest way to check this is to open in a web browser the following URL: http://localhost:8080/elarp/pimpserver which will list all available commands.

The PIMP servlet currently accepts commands via URLs in the following form:
http://host:port/elarp/pimpserver?exec=<c>cmd</c><p>param1</p><p>param 2</p> ...<p>paramN</p>

Such URLs can for instance be sent to the servlet via an ordinary web browser, simply type the URL in the address field and press Enter. For more details on how to use this, see the example scenario at the end of this chapter.

9.8 Simple text logger

All clients running on a PC require a network_properties.xml file to set the network configuration. In the distribution provides such a file, with a default configuration for clients to connect to the same machine as they are started on. The file looks like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>Network settings for PIMP.</comment>
  <entry key="ServerHost">localhost</entry>
  <entry key="ServerPort">1000</entry>
  <entry key="ClientPort">4242</entry>
</properties>
```

If you are running your session server on another machine, please change localhost to the address of that machine, either its DNS name or IP-address. If you want the session server and consequently the clients to use another port, change ServerPort. ClientPort is not currently used by the PIMP system but would typically allow two Pimp process to connect directly to each other using this port.

To start a simple text logger, locate the file start_textlogger.bat in the run/ directory and double click it to start. A new window shall open up and you should see a text output similar to:

```
D:\home\kalle\momentum\run>java -cp ..\momentum_j2se.jar;..\lib\pimp_j2se.jar;..\lib\mogile_j2se.jar;..\lib\part_j2se.jar org.iperg.pimp.utils.SimpleTextLogger
PimpLoggerManager: Connecting to server using url tcp://localhost:1000
PimpLoggerManager: Connection to 'tcp://localhost:1000' was successful
```

If you look at the Session Server window you will see that the logger also connected to it:

```
SessionServer: Connection established from 1158675856640:0:1:0
SessionServer: New configurator connecting 1158675856640:0:1:0
```

9.9 PocketPC

It is also possible to run the phone client on a PocketPC as well, though this might be a bit trickier.

**Requirements:**

You need a PocketPC, currently (2006-05-15) the software has only been tested with a Dell Axim X51v with PocketPC Mobile 2005.
To run a PIMP client on a PocketPC there are a number of prerequisites. You need a Java Virtual Machine with Bluetooth support. Several Pocket PCs are shipped with a Java Micro Edition runtime environment though none of these have support for Bluetooth as far as we are aware. You need to setup a working Java Virtual Machine runtime and add Bluetooth support. As explained in the following section this will include the: installation and setup of a Java Virtual Machine (JVM) runtime, add Bluetooth support, create shortcuts for quick and easy start of the application.

9.9.1 Runtime Installation

The JVM runtime we have used is called Websphere Everyplace Micro Environment, also known as IBM J9. It may be downloaded from IBM through their trials and betas section. At this site http://www-306.ibm.com/software/wireless/weme/ tin the menu on the left you will find Trials and Betas. Under this section you will find different versions to download.

The following two versions have been used on the Axim X51v.

CLDC 1.1/MIDP 2.0 for Windows Mobile 5.0/ARM
CLDC 1.1/MIDP 2.0 VGA (Hi-Res) for Windows Mobile 5.0/ARM

The following runs also on the Axim but we have not tested the Bluetooth functionality with this one yet.

CDC 1.1/Foundation 1.1/Personal Profile 1.1 for Windows Mobile 5.0/ARM

Download either one of the MIDP2.0 versions of the Websphere Everyplace Micro Environment.

The installation instructions on the IBM website are not very good for the different packages. Follow these instructions instead:

1. Run the installation you downloaded, e.g. ibm-weme-wm50-arm-vga-midp20-6.0.0-20060321-073252-168.exe. This will extract the runtime files into C:\Program Files\IBM\WEME\runtimes\61\wm50-arm-vga-midp20/.

2. Extract the PocketPC runtime from the runtime installation. The runtime is in a zip-file. You will find this zip-file in the installation directory. In the version we downloaded the file was called weme-wm50-arm-hires-midp20_6.1.0.20060317-111429.zip Extract this into a directory on your hard drive.

3. On your PocketPC create a directory called J9. We recommend you to create this directory in the device root which is the directory used in the instructions below. If you chose another directory make sure you change all paths like in the shortcut setup to reflect this change.

4. In the J9 directory create a directory called MIDP20.

5. Copy all the files extracted from the zip-file into MIDP20. Now you have J9 JVM installed on your PocketPC.

9.9.2 Bluetooth support

To achieve Bluetooth support in the Java Runtime we have been using the Avetana implementation of JSR-82. JSR-82 is the Java API specification for Bluetooth on mobile devices like mobile phones or Pocket PCs. You can either buy or download a
trial version of the Avetana JSR-82 implementation. Though to get hold of the right version of the Avetana implementation is not necessarily so easily done. The problem is that the available test version on the Avetana website does not work with the Dell Axim X51V and the MIDP version of the Websphere Everyplace Micro Environment.

The following section will try to help you as much as possible to make this easy. Unfortunately Avetana is the only option we have found to get Bluetooth support on a PocketPC.

9.9.2.1 Get a working version

What you need to do is either download a demo version of their JSR-82 implementation or buy one. Once you have the jar-file you need to extract the licence from it and place it in the jar provided in this release. Extract the license.txt file from avetanaBluetooth.jar which you downloaded and place it into the jar-file avetanaBluetooth-midp.jar which is found in Avetana-JSR-82_for_j9midp.zip This is the recommended procedure by Avetana.

If you are using another PocketPC device the version available on the Avetana website might work. So far we have only tested our software on the Dell Axim so we cannot guarantee this is working on other devices. Either buy or download the trial version of Avetana JSR-82 implementation. Available through their website at http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml

9.9.2.2 Installation

In the zip-file (Avetana-JSR-82_for_j9midp.zip) from Avetana there is an installation instruction called INSTALL. When following this instructions it is important to know which Bluetooth stack your device uses. The Axim X51v that we have tested on uses the Microsoft Bluetooth stack. Use the avetanaBTCE.dll from the directory ce42/Microsoft/ on this device as instructed in the installation.

9.9.3 Software installation

The application is distributed as a MIDP application. MIDP applications are distributed both as a jar-file which includes the actual software and a jad-file that includes information about the software, required when launching the application.

You can install both the momentum_j2me.jar and momentum_j2me.jad wherever you want on your device, though we recommend you to place these files in the root directory of the device, the topmost directory. If you install in the root directory you do not need to change any paths when following these instructions.

9.9.4 Shortcut setup

Currently there exists only one Momentum client application which is simply called Momentum. The most convenient way to start the Momentum application is to create a shortcut on the PocketPC. By doing so you just need to tap on the icon for the shortcut to start the application- If you do not create a shortcut you need to use the console within the Java Runtime to start the application.

To setup a shortcut follow these steps:

1. Create a text file on your PC, e.g. Momentum.txt.
2. Open the file and type in the following line:
255#"\J9\MIDP20\bin\j9.exe" "-Xmjit" "-jcl:midp20"
"-Xbootclasspath:\J9\MIDP20\lib\jclMidp20\jclMidp20.jxe" "-cp"
"\momentum_j2me.jar" "javax.microedition.lcdui.AppManager"
"momentum_j2me.jad"

If you have installed the Java Runtime in some other directory than \J9\MIDP20
or put the momentum_j2me.jar and momentum_j2me.jad in some other directory
you need to change the paths in the above text to reflect those changes.
\J9\MIDP20 in the above example is where the JVM is installed. And the
momentum_j2me.jar is stored in the device root \\.

3. Rename this file to Momentum.lnk and copy it to your device. We usually place
it in the device root but essentially it could be placed anywhere on the device.
Once you copied over the file to your device you should be able to start the
application by clicking the Momentum icon in the device root or wherever you
installed it.

9.9.5 Starting the application

To start the application click on the shortcut you just created. Once started you should
see the application start-up screen. Now you can follow the instructions as stated in
Phone client section, only difference being that you use the pen instead of the phone
keyboard for interaction.
10 Appendix A

This appendix includes a drawing giving a glimpse of the complexity of the inner workings of the pimp objects and their properties, in part of two processes in the Momentum setup ("Node Server" was the working name of the Game Server process).
Figure 9 All the different PimpObjects in the game system.