DEGREE PROJECT AT CSC, KTH

Development of a web based interface for game-masters of pervasive games

Guerrero Corbi, Víctor
KTH e-mail: victorgc@kth.se
Degree project in: Computer Science
KTH Supervisor: Höök, Kristina
Mobile Life Supervisors: Waern, Annika and Back, Jon
Examiner: Bälter, Olof
Project provider: Mobile Life

June, 2014
Development of a web based interface for game-masters of pervasive games

Abstract
Pervasive games are games played in the real world with various support from mobile and sensor technology. It is becoming much simpler to commercially stage pervasive games due to the high level of penetration of smart mobile phones. However, technology supported pervasive games are seldom fully automated games. There are two main reasons for this phenomenon, many of the engaging game activities such as the physical encounter and the interaction between players cannot be fully sensed and captured by technology. Secondly, the physical world wherein pervasive games are staged provides an ever-changing scene, a scene that interacts and interferes with pre-programmed functionalities. For these reasons, many pervasive games require game-mastering functions.

The goal of this project is to study the interfaces and systems to provide the game-mastering functions in the context of pervasive games using the game Codename: Heroes as its use case. As a result we present a generic Service Oriented Architecture for game-mastering interfaces of pervasive games, and a semi-generic interface that uses a visualization tool that combines the spatial, temporal and social domains. The visualization tool explores the manipulation of time-ranges in a timeline to facilitate the exploration of the time domain.
Utveckling av ett webbaserat gränssnitt för spelledare i ubika spel

Sammanfattning


# Table of Contents

Introduction ........................................................................................................................................... 1  
1.1 Problem formulation .................................................................................................................. 1  
1.2 Delimitations ............................................................................................................................ 2  
1.3 Objectives ................................................................................................................................... 2  
1.4 Research approach ................................................................................................................... 2  
1.5 Thesis overview ......................................................................................................................... 4  

## I Background ................................................................................................................................. 5

Games .................................................................................................................................................... 6  
2.1 Pervasive Games ....................................................................................................................... 6  
2.2 Game-mastered pervasive games .............................................................................................. 8  
2.3 Technology in pervasive games ............................................................................................... 9  
2.4 Summary ..................................................................................................................................... 9  

Space-time domain .................................................................................................................................. 11  
3.1 Data models .................................................................................................................................. 11  
3.2 Visualization techniques .......................................................................................................... 12  
3.3 Space-time visualization in pervasive games ........................................................................... 13  
3.4 Summary ..................................................................................................................................... 15  

Distributed system communication ......................................................................................................... 16  
4.1 Web services ............................................................................................................................ 16  
4.2 Real-time communication ....................................................................................................... 19  
4.3 Summary ..................................................................................................................................... 20  

## II Methodology .................................................................................................................................... 22

Analysis .................................................................................................................................................. 23  
5.1 Codename: Heroes ..................................................................................................................... 23  
5.2 Functional requirements ........................................................................................................... 25  
5.3 Non-functional requirements ..................................................................................................... 29  

Architecture ............................................................................................................................................. 30  
6.1 Game-master interface ............................................................................................................. 30  
6.2 Logger system ............................................................................................................................ 33  
6.3 Annotation system .................................................................................................................... 36  
6.4 Game engine ............................................................................................................................. 36  

Implementation ...................................................................................................................................... 37  
7.1 Game system .............................................................................................................................. 37  
7.2 “Pervastered”, game-master interface ...................................................................................... 42  
7.3 “Logee”, logger system ............................................................................................................. 57  
7.4 Disqus, annotation system ......................................................................................................... 60  

Evaluation ............................................................................................................................................... 61  
8.1 Interface ....................................................................................................................................... 61  
8.2 System ......................................................................................................................................... 64  

## III Discussion ..................................................................................................................................... 71

Conclusions and future work ............................................................................................................... 72  
9.1 Discussion .................................................................................................................................... 72  
9.2 Future work ................................................................................................................................... 73
List of tables
Table 1: Examples of different space-time models depending on the approach .......................... 12
Table 2: HTTP verbs usage in REST APIs .............................................................................. 18
Table 3: HTTP code usage in REST APIs .............................................................................. 18
Table 4: MOO API endpoints ................................................................................................. 38
Table 5: Usefulness of transmitted data .................................................................................. 39
Table 6: Real-time API endpoints ......................................................................................... 42
Table 7: Requirement compliance of different existing web-services ................................. 57
Table 8: Top memory and processing consumption processes in the logger system ............. 64
Table 9: Metrics resulting of averaging the play-sessions weighted by play-time ................. 67
Table 10: Events generated with an arbitrary play-time of 1 hour per day ......................... 68
Table 11: Top memory and processing consumption processes in the game system .......... 68
Table 12: Elements and total size of each resource of Codename:Heroes ......................... 69

List of figures
Figure 1: Small multiples visualization technique ............................................................. 13
Figure 2: Space-Time cube visualization ............................................................................. 13
Figure 3: Wakame technique ............................................................................................... 13
Figure 4: Great Wall visualization technique ....................................................................... 13
Figure 5: Visualization of space time with paths in CatchBob! ........................................ 14
Figure 6: STC visualization in PAC-LAN ........................................................................... 15
Figure 7: Service Oriented Architecture elements and relationships ................................. 16
Figure 8: SOAP message expressed in XML ...................................................................... 17
Figure 9: Polling strategy ..................................................................................................... 19
Figure 10: Push strategy ....................................................................................................... 20
Figure 11: Codename: Heroes class diagram of main entities ........................................... 24
Figure 12: Authentication use cases ..................................................................................... 27
Figure 13: Player uses cases ............................................................................................... 28
Figure 14: Rituals use cases ............................................................................................... 28
Figure 15: Messages use cases ............................................................................................ 28
Figure 16: Quests use cases ............................................................................................... 29
Figure 17: Systems and relationships of the proposed solution ........................................ 30
Figure 18: Thin-client architecture ..................................................................................... 31
Figure 19: Thick-client architecture .................................................................................. 32
Figure 20: Class diagram of the Logger System .................................................................. 33
Figure 21: Sequence diagram of the operation Updater::update ....................................... 35
Figure 22: Class diagram of the Annotation System .......................................................... 36
Figure 23: Communication between the systems ................................................................. 37
Figure 24: Game system specification .................................................................................. 37
Figure 25: Horizontal scaling available with a game API as an entry point to the game engine 38
Figure 26: API documentation in Apiary.IO ........................................................................ 39
Figure 27: Example of publisher-subscriber pattern components ...................................... 41
Figure 28: data-flow in EmberJS ...................................................................................... 44
Figure 29: Model layer class diagram ................................................................................. 44
Figure 30: Controller layer class diagram .......................................................................... 45
Figure 31: View layer class diagram .................................................................................. 46
Figure 32: Structure of the web application ....................................................................... 46
Figure 33: Summary HTML view ...................................................................................... 47
Figure 34: Full view HTML ............................................................................................... 47
Figure 35: Quest visualization ........................................................................................... 48
Figure 36: Players in each sub-quest can be seen in the nodes .......................................... 48
Figure 37: Sankey diagram to visualize accumulative flow of players .............................. 49
Figure 38: Marker clustering technique .......................................................................... 50
Figure 39: Timeline HTML view ....................................................................................... 51
Figure 40: Time range manipulation keeping length ........................................................ 51
Figure 41: Events in timeline ............................................................................................ 52
Figure 42: TardisJS architecture ...................................................................................... 53
Figure 43: Visualization of paths in the map ..................................................................... 53
Figure 44: Time range mapped in space using circles ....................................................... 54
Figure 45: Time range with 3 steps .................................................................................. 55
Figure 47: Heat-map visualization .................................................................................... 55
Figure 48: Path difficult to read at a glance ..................................................................... 56
Figure 49: Path aided with heat-map to help visualization ............................................... 56
Figure 50: Updater edition HTML view .......................................................................... 58
Figure 51: Error shown in red .......................................................................................... 59
Figure 52: format of filter petition .................................................................................. 59
Figure 53: format of the generated events ....................................................................... 59
Figure 54: Disqus box ..................................................................................................... 60
Figure 55: CPU load of Python process ........................................................................... 65
Figure 56: CPU load of Mysqld process ........................................................................... 65
Figure 57: CPU load of Nginx process ............................................................................. 66
Figure 58: Response time of event consumption ............................................................. 66
Figure 59: response time for requests to the Logee API zoom in lower bound .................. 67
Figure 60: CPU load for Python process zoom in lower bound ........................................ 67
Figure 61: CPU load of Node process ................................................................................ 69
Figure 62: CPU load of Moo process ............................................................................... 69
Figure 63: Response time of MOO API based on resource and simultaneous users .......... 70
Chapter 1
Introduction

Pervasive games are games played in the real world with various support from mobile and sensor technology [1]. It is becoming much simpler to commercially stage pervasive games due to the high level of penetration of smart mobile phones. Successful commercial examples include games like Google’s Ingress\(^1\) and Shadow Cities\(^2\).

Technology supported pervasive games are seldom fully automated. There are two main reasons for this phenomenon, first; many of the engaging game activities, such as the physical encounter and the interaction between players cannot be fully sensed and captured by technology. Secondly, the physical world wherein pervasive games are staged provides an ever-changing scene, a scene that interacts and interferes with pre-programmed functionalities. For these reasons, many pervasive games require game-mastering functions [1]. The game-master might hereby be an individual or a group of people who works behind the scenes to modify and moderate a running game.

1.1 Problem formulation

The fact that pervasive games can take place anywhere in the real world makes it difficult for the game-master to constantly be present at the players’ location. Hence, it is of vital importance that we consider from the start a solid understanding of how game-masters can be provided with systems that facilitate the task of information overview and management. Previous studies of game-mastered pervasive games have focussed on the functional requirements for the game-mastering task [1], in-game authoring tools [2, 3] and methods for gathering information about player activities [4, 6]. Even though these studies have included the development of game-mastering interfaces, the reason behind their development was to provide functionalities that were necessary to conduct the main topic of research, which was not the game-master’s interface per se. Thereby, leaving the game-mastering interfaces out of the main focus of study and resulting in some insights about the interfaces developed [1, 2, 4, 6] but few studies have focussed in describing the development process, architectures or technologies used to build the systems [5]. An effective game-mastering interface is highly dependent on each game [6]. Accepting the previous premise, the research goal of this thesis project is to develop a generic architecture for interfaces of game-masters of pervasive games that allows adaptability.

To reach this goal we study game-mastering interfaces from the perspective of Information Systems (IS) and approach it from these perspectives:

- **Visualization of information gathered from player activity**: Pervasive games are defined by Montola as “games that expand the contractual magic circle of play: socially, spatially or temporally” [7]. The space, time and social domains need to be visualized in a way that allows the game-master to have an understanding that ranges from a global overview to a detailed understanding of each player’s actions [6]. However, the challenge in visualising these data sources lies in their combination [8]. We use maps to visualise spatial information with solutions like Google Maps and Bing Maps. On the other hand, social networks have presented ways to visualize social events in timelines, such as the Facebook timeline. To approach the problem of the visualization of these domains, Chapter 3 presents data models and visualization techniques of the space and time domains which later on, in Section 7.2.4 are used to design a new web-based visualization technique to facilitate the understanding of the information domains of pervasive games.

\(^1\) http://www.ingress.com/
\(^2\) http://www.shadowcites.com/
• **Distributed system integration**: Interoperability of various systems raises concerns about their communication. Game-mastering systems face this challenge as the system has to collect data from different sensors, and there is the game-engine itself, which can be a separate system from the game-mastering system. In Chapter 6, we present the use of a Service Oriented Architecture to approach the creation of the game-master interface system.

• **Reusable architectures**: Game-mastering interfaces have been typically built from scratch for each of the pervasive games that needed it. One of the goals of this project has been to develop a modular architecture that facilitates reusable components. The separation of concerns allows flexibility in the adaptation of the architecture to each game to respond to the fact that the creation of effective interfaces for game-mastering is highly tied to each game [6]. The modular approach can speed up the development of future game-mastering interfaces as it proposes the set of necessary modules needed for game-mastering interfaces of pervasive games. In Chapter 6 we identify the modules from the game-mastering functionalities found in Chapter 5. Along with the modules, in Chapter 7 we analyse web-services available in the market that provide functionalities needed by each module (Section 7.4) and study the creation of web-services to cover the functionalities that are not covered yet (Section 7.1, Section 7.2 and Section 7.3).

### 1.2 Delimitations

The development of the generic interface has been done using the pervasive game *Codename:Heroes* but its evaluation in multiple games is out of the scope of the project. All the sub-systems created to support the main functionalities have been developed to provide the basic functionalities needed for the interface to work.

### 1.3 Objectives

The project’s main goal is to study the *tools that aid the game-mastering function in the context of pervasive games* including not only the final tools but also the systems that are used to provide such tools. Thus being of interest the architectural and technological level as well as the usability of the delivered tools. To accomplish these objectives, the project includes a specification, implementation and evaluation of a web-based game-mastering interface using the context of the pervasive game *Codename: Heroes* as its use case.

### 1.4 Research approach

The research approach used for the work done in the thesis is the **Design Science in Information Systems Research** (DSR) framework presented in Hevner et al. 2004 [9]. The framework proposes the creation and evaluation of artifacts to solve identified organisational problems. The artifact generates knowledge as it can be adapted as necessary by other professionals to fit their problem. In this project we treat the game-mastering system as an IS separated, yet dependent on the information system of the game itself. We have chosen DSR as it is a validated approach for the study of Information Systems which are the main focus of this project.

Hevner et al. 2004 proposed seven guidelines that can be used to evaluate good design science research in IS:

• **Design as an artifact**: DSR must include the production of an artifact through means of *constructs* (vocabulary and representations), *models* (abstractions and representations), *methods* (algorithms and practices) or *instantiations* (implementations and prototype systems).
Problem relevance: The objective of DSR is to create solutions to important business problems using technology.

Design evaluation: The artifact must be proved using evaluation methods.

Research contributions: DSR must provide clear and verifiable contributions to design methodologies, design foundations or design of artifacts.

Research rigor: Rigorous methods have to be applied in the construction and evaluation of the artifact.

Design as a search process: The production of an effective artifact follows an iterative process that can use available artifacts to reach partially satisfied end goals.

Communication of research: The research must be presented to management and technology oriented audiences.

The guidelines apply to this project as follows:

Design as an artifact: We have produced an instantiation of a game-mastering interface.

Problem relevance: Effective game-mastering systems are highly dependent on games [6] yet there are few studies focused in IS for game-masters of pervasive games. In the last years, we have seen the staging of pervasive games increase out of the scope of research with games like Google’s Ingress. The motivation behind this increase has been the accessibility of developers to the sensors in smart mobile phones available to the public. To further help the development of game-mastered pervasive games in research and the market it is important to provide tools and knowledge that aids its development. In this case, we approach the creation of tools for game-masters rather than in the game engines.

Design evaluation: The game-master interface is validated against the requirement list at a technical and usability levels.

Research contributions: In the thesis we extract a requirement list (Chapter 5) for the pervasive game Codename: Heroes using as base the game-mastering functions described in [1]. Using the requirement list we build a specification of a generic service-oriented architecture in Chapter 7 that can be adapted to fit other pervasive games as each of the services full-fills a subset of the requirements. We provide considerations to the communication between the subsystems and the creation of Application Programming Interfaces (APIs) for pervasive games and logging systems. Finally, the report contains in Subsection 7.2.4 a new visualization tool to interact with space-time based on the manipulation of time ranges in a timeline.

Research rigor: The production of the artifact uses proven software development process and design patterns to adhere to good-practices and grounded results. The artifacts created in each of the steps of the development use standardized methods and languages used within the field of software engineering: Unified Modelling Language, Class diagrams and Sequence Diagrams.

Design as a search process: The development of the artifact has followed an iterative process along with the game-master team of Codename: Heroes to provide feedback and validate each iteration of the development.

Communication of research: This report presents the development process and the context in which it has been applied explaining technological terms and tools to an audience that ranges from the technological professionals to game-masters. The requirement analysis, architecture and the implementation details are discussed in this report, and all the code is available online.

https://github.com/codename-heroes/pervastered-interface
1.5 Thesis overview

The thesis has three sections: background, methodology and discussion. First of all, the background literature in Section I grounds the different domains covered in the thesis. Chapter 2 introduces the reader to game’s definition and develops it towards pervasive games and the figure of the game-master. Following in Chapter 3 is the explanation of data models and visualization techniques of the domains covered by pervasive games: space and time. To end up the background section, Chapter 4 presents different service oriented architectures to approach the communication of distributed systems.

Later on, Section II describes with details the different steps of the methodology: starting in Chapter 5 with the analysis of the requirements based on the game-mastering functions, followed by Chapter 6 specifying the system’s architecture. Afterwards, Chapter 7 presents the implementation of the proposed system architecture along with a combination of timeline and map visualization tool to cover the visualization of the space and time domains. The section concludes with Chapter 8 presenting an evaluation and analysis of the proposed solution and overall system suitability.

Finally, discussion in Section III sums up the results obtained and lists future work that can be addressed after this project.
Designing interfaces for pervasive games involves several fields of study. In this section we present the main topics needed to understand the domains covered within this thesis project as well as base knowledge used afterwards to ground the decisions made during the development process. In the beginning we introduce game-mastered pervasive games, starting by viewing games in general and following the evolution of its definition to understand the framing of pervasive games. By an attempt to cover the creation of interfaces for pervasive games it is necessary to understand the domains they deal with, space, time and social. We present different data models and visualization techniques used for these domains. Later on, they are contextualized using existing game-mastering interfaces of pervasive games. Finally, an introduction to distributed system communication to cover the theory behind the communications of the web applications and services that have been developed for the game-mastering IS.
Chapter 2
Games

To comprehend games we have to understand what their definition is. However, there is no formal agreement in the definition of games [10], hence instead of looking at different existing ones we are just focusing on the ones that are necessary to understand the approach of the definition of pervasive games that is used in the project.

Johan Huizinga, considered by many the father of game studies, explains games in terms of play, where games are a ritual activity outside ordinary life within particular boundaries of time and space that it is reigned by a set of fixed rules [11]. Later on, the focus in defining it as an activity outside the ordinary life brought Katie Salen and Eric Zimmerman [12] to frame play using Huizinga’s metaphor of the magic circle, the boundary that separates ordinary from ludic space. This new framing allows them to focus on the separation of spaces itself, and differ from Huizinga in that play does not need to happen in a certain specialized area at a particular time.

As an example of this separation Salen and Zimmerman propose an online chess match where space and time are not constrained. They define games as systems (as compared to Huizinga’s definition of game as an activity) where players engage through an artificial conflict defined by rules that result in a quantifiable outcome [12]. Even though this definition abstracts games from particular space and time boundaries it still contemplates the separation of ludic and ordinary space. Pervasive games like Killer [13] challenged this separation of ludic and ordinary. The game is played in the real world and the goal of players is to assassinate other players using imaginary weapons. Killer challenges Salen and Zimmerman’s definition blurring the separation of ludic and ordinary allowing any location and time to be valid to perform an assassination. Also, as players are not aware of all the other players, but just the ones they have to assassinate; people nearby could potentially be their assassin. This unclear barrier breaks any possibility of separation between ludic and ordinary; avoiding specific play sessions and embracing any space as part of the game. Hence, pervasive games break the space and time boundaries of the magic circle blurring the separation between the ludic and ordinary.

2.1 Pervasive Games

As for the different definitions of game, the same occurs with the definition of pervasive games [14]. Therefore before stepping into the formal definition of pervasive games that this project uses, it is useful to check the meaning of the adjective pervasive: “(especially of an unwelcome influence or physical effect) spreading widely throughout an area or a group of people”4. A remark of this definition is the emphasis not just in the widespread, but also the fact that its influence is uncontrollable. Markus Montola defines pervasive games as “games that have some features that expand the contractual magic circle of play spatially, temporally, or socially” [15].

Spatial expansion refers to mixing the ludic with the ordinary space embracing players’ surroundings and context instead of isolating from them. Space in this context does not just refer to physical architecture, but also to properties of the physical world that can become part of the game when players decide to include them. As an example, in Killer players can use any item as a weapon. For instance, a camera with an objective can be used as a sniper weapon while other imaginative inclusions can be using an orange as a hand-grenade or a banana as a pistol. The spatial expansion in Killer embraces any inclusion from the ordinary into the ludic space, and it is up to players to imagine and decide what they want to use as a weapon.

4 http://www.google.com built-in dictionary
**Time expansion** suppresses play sessions blurring the difference between ludic and ordinary. Salen and Zimmerman’s game definition breaks the boundaries of specific space and time boundaries explaining play intervals in terms of play sessions. This consideration takes into account the understanding of game as a consent action by players to engage the game, and that at some point, can decide to take breaks in which game rules do not apply. In *Killer*, there is no differentiation between ludic and ordinary time. Once players agree to join the game, there is no common break time, and any time is part of the game. Even if a player is doing homework in the library, another player could be playing nearby and try to kill him. Hence, the concept of time of play is uncertain and difficult to define since there are no global play-time breaks.

Finally, and as a consequence of mixing the ludic and ordinary space, pervasive games can make non-players participate in the game and, therefore, pervasive games have a **social expansion**. The participation of external people has to be taken into account when designing the games and has to deal with possible problems raised by involving them. As examples of different non-voluntary player participation in games; in the mentioned *Killer*, non-players are treated as mere spectators. In order to avoid misunderstandings with non-players thinking that an in-game assassination is real, the game penalizes assassinations with witnesses. Opposed to *Killer*, in the game *Cruel 2 B kind* the designers tried to involve non-players by changing *Killer*’s violent assassinations for kind actions while keeping the core mechanic. Thus, if a player performs one of the predetermined kind actions to another player, the targeted player loses. Another addition to *Cruel 2 B kind* is that players do not know their targets and, therefore, they have to guess who is playing. This mechanic can cause that players end up performing the stipulated kind actions to non-players, in which case nothing happens to the targeted person. The other role that non-players can take in *Cruel 2 B kind* is as killers, which might casually happen if non-players perform one of the kind actions to the player.

Different grades of each expansion lead to different experiences, and thereby there is a rich group of **subgenres in pervasive games**. At the same time, each subgenre can contain games with different levels of each expansion. In **treasure hunts**, players try to find objects hidden in the real world. One example of this subgenre that focus on spatial and temporal expansion is *The Game*, in which players play continuously in the ordinary world solving puzzles in order to know where the next puzzle is, and with the ultimate goal of reaching the final prize. On the other hand, *Insectopia* [88] has a greater social expansion using non-player’s Bluetooth devices to generate unique insects that players have to find and capture using their mobile devices. Another subgenre is **Assassination games** emerged from the film *La decimal vitta* (1965). This subgenre includes games where players receive information about the player they have to exterminate without knowing who has to exterminate them. As players do not know who their assassin is, they have to be cautious with people around them. Examples in this subgenre include the aforementioned *Killer* where players can use any object from real life to simulate a fake weapon: carrots representing knives, umbrellas as rifles. Other examples include *Cruel 2 B Kind* and *BotFighters* [25] where players have a virtual robot that they can use to attack the robots of other players that are nearby in the physical world. More recent subgenres include **pervasive LARPs** that brought the genre of living action role-playing games (LARP) to the ordinary world with games based on *Vampire: The Masquerade*. A role-playing game where players perform as vampires with unique powers. To conclude the list of subgenres, **alternate reality games (ARG)** make use of current technology to recreate and bring a fictional story to life and make it evolve according to players’ actions. Here, players perform as themselves and pretend that they are in real life as in contrast to pervasive LARPs where players play a fictive role. *The Beast* [16] was the game that started the subgenre of ARGs and focused around a mystery of a complex murder playing in the year 2142. The game engaged three million players world-wide using the fictional world of Steven Spielberg’s film *A.I.: Artificial Intelligence* to build the story as players solved the different puzzles and stories. Other recent examples of ARGs include Google’s *Ingress* which explains the story of an unknown group force that wants to change mankind using mind control devices. Players choose to join one of the two existing factions, to either help or fight the unknown group force by controlling the virtual mind controllers that are positioned in the real world locations. Players can interact with the virtual
world using mobile phones while moving in the real world as the virtual world presents a layer of interaction over the real world.

Even though some pervasive games are fully automated, this is not always the case as we have seen in The Beast or, to list some other examples, Epidemic Menace or Uncle Roy all around you. Some of the reasons which explain the necessity of the game-master are that current technology has been insufficient to track players’ actions and the poor ability of current fully automated systems to respond to gameplay that might arise, and that is not fully or partly contemplated.

2.2 Game-mastered pervasive games

The term game-master was first introduced in the world of table-top role-playing games to act as the role that decided game logic and maintained a storyline based on players’ actions. Moving the control of the game logic to a human allowed for more unpredictable and elastic answers to players’ improvisations as compared to the logic applied by artificial intelligences. It allows players more freedom in their decisions and potentiates emergent gameplay that is not necessarily contemplated by the game designer. In pervasive games, emergent gameplay is encountered as it contains elements with infinite affordances. Jane McGinigal used the concept of infinite affordances to explain the concept of player’s movements in pervasive games. She uses Gibson’s concept of affordance, “an action possibility available in the environment to an individual”, to reflect the fact that every element is a potential game element giving the possibility to infinite variations of movements at any point. Pervasive games can benefit from game-mastering and they have used active game management through game-masters to:

- **Maintain the illusion of the game world**: Automated systems cannot properly respond to infinite affordances, but game-masters can respond better to emergent gameplay as they can modify the game as needed to deliver a more realistic game world.

- **Adapt the difficulty level**: Editing and authoring contents for the game while it is running allows game-masters to suit the difficulty level depending on players. In The Beast, game-masters were not initially aware of the combined capabilities of their players. With more than three million players solving the puzzles together, their knowledge enormously exceeded the expectations of the solving time for the first puzzles leading to the addition of posterior ones. On the other hand, sometimes, game-masters can also sub-estimate players’ capabilities. In Momentum, players were divided into groups. Soon after starting playing, the game-masters realised that not all players had the same level of implication. As a result, game-masters had to create content that suited the pace of both slower and faster teams. Otherwise the level would have been inadequate for all the teams and ultimately would have affected the overall enjoyment of the game.

- **Game steering**: Even though players are free to act, in some pervasive games with interactive stories, game-masters have thought of a set of possible paths. It is their responsibility to interpret players’ actions and to lead them through one of the pre-existing paths. Game-masters can actively modify the content, and with that, try to steer players into a particular storyline. An example can be seen in the game-masters of the game Monitor Celestra, a LARP based on the world of the science fiction TV show Battlestar Galactica. Game-masters structured the game in several days and let players decide freely how they wanted to spend one of the days. However, even though during that day game-masters allowed players to explore the galaxy, establish combat with enemies or just stay in the spaceship; the overall story had already been planned. Game-masters had to steer the story towards one of the possible endings.

---


• **Human backup system**: Game-masters can respond to technology failures repairing or substituting the malfunctioning systems. As they have knowledge of the game-state, if it is necessary to respond to a malfunction that cannot be fixed immediately, they can try to create the illusion that there is no such error. For example, by manually creating and sending messages replacing a computerized system that is not working.

### 2.3 Technology in pervasive games

As we have seen, some of the mentioned pervasive games such as *Cruel 2 B Kind* and *Killer* do not use technology, but they can certainly benefit from it. To mention some of the possibilities, mobile phones can aid communication between separated players, a game engine can control game logic, and positioning technologies can help keeping control of players’ positions. But using technology also provides some inconveniences as it can be expensive, and its reliability is not always perfect. Therefore, it is important for game-masters to take into account the limitations of the chosen technology. Matthew Chalmers defines **seamful design** [23] as a design approach that makes technology inner workings visible to its users. In that sense, he embraces not just taking technology limitations into account but using them as part of the game. In *Treasure* [24], players use Global Positioning System (GPS) and Wireless Local Area Network (WLAN) to play the game. Players have to interact with virtual coins that can be picked when a player is near the position of a coin. Once players pick up a coin, they have to deliver the coin to their own treasure which is just accessible if the player has WLANsignal. While a player is not in WLANcoverage, another player nearby can steal her coins. In order to master the game, players have to understand WLANand GPSareas of coverage and use them to their own benefit.

Technology can be approached by pervasive games due to its **practicality** and, in that case, we can find two approaches: **technology-sustained** and **technology-aided** games. On one hand, technology-sustained games rely on computers maintaining the game state and altering it through reaction to players’ actions. Similar to computer games but with a physical world interface. In *BotFighters*, players can attack other players’ robots if they are physically nearby. The game server that simulates the game maintains the battle simulation and position of all players. On the other hand, technology-aided games use technology to support some, but not all game activities. In *Momentum*, technology was used as a medium for staging and communicating the game. Players had to do rituals in different locations, and microphones installed in a technologically empowered doll recorded how players performed the rituals. Then game-masters could evaluate the performance using a game-mastering interface that let them access the stored ritual performance. Even though *Momentum* was heavily technology dependant it did not entirely rely on it.

Technology can also be used for **aesthetical reasons**. Different emotions can be conveyed through different interactions and reactions with technology. Reeves et al. [26] provides a mapping of how technology can be perceived depending on its manipulation. People can manipulate technology in a non-revealing way (a player hiddenly presses a button in a magic doll) or more exibitory (performing gestures), and the feedback of that interaction can also be hidden or revealed.

### 2.4 Summary

**Pervasive games** are games that expand the magic circle of play spatial, temporal or socially. Game designers use different levels of these expansions to create different subgenres of pervasive games.

The figure of the **game-master** can provide better reactions to the infinite affordances in pervasive games.
Technology can be used to give full support to the game, aiding monitoring tasks and controlling game logic or it can be used partially for its aesthetical component when interacting with it.
Chapter 3
Space-time domain

The main data domains in pervasive games as stated by Markus Montola are space, time and social [15]. To aid the design decisions that we are going to take afterwards, in this chapter we present the different data models and visualization techniques used for the space-time domain.

3.1 Data models

The oldest representation of spatial data found is in the form of a scratched map on a mammoth bone dated from more than 15,000 years ago [8]. Humans have been using this representation since the very beginning, and it has evolved towards a more dynamic approach, being the first computer map created by Waldo Tobler in the late 1950s [27]. Computer maps kept evolving towards more complex systems: Geographic Information Systems.

The first Geographic Information System (GIS) was created in Ontario in 1962 and along with Database Management Systems (DBMS) are the two systems that end up allowing storage and manipulation of spatial-time data. In 1980s, DBMS were provided with time operations, which had been propitiated by the rapid decrease of storage cost [8]. This possibility opened discussions in modelling of time-dependant spatial data covered by [28, 29] and led to the snapshot model [28] which gathers data of the whole system at each timestamp and generates a layer with it. GIS have been using layers as a technique to allow manipulation and visualization of different data types, the approach in this model uses ordered layers to separate spatial data along time. In 1994, Peuquet [30] developed the Triad model. She stated that in order to understand spatial-temporal data, three questions must be answered: where (space), when (time) and what (objects). With these elements, she concludes that the three questions that must be answered when exploring spatial-time data [30] are:

- When + where → what: detect objects that where at a particular position and time
- When + what → where: detect objects’ location at a particular time
- Where + what → when: detect the time that an object or group of objects were at a particular location

In Table 1, we can see how different approaches towards space, time and objects allow a classification of different data models.
Table 1: Examples of different space-time models depending on the approach

<table>
<thead>
<tr>
<th>Approach</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space</strong></td>
<td><strong>Location based</strong> [31]: Raster and vector representations of space.</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td><strong>Snapshot</strong> [28]: At each timestamp the system stores the state of all the world.</td>
</tr>
<tr>
<td></td>
<td><strong>Event based</strong> [32]: Stores data changes in time. Hence, to visualize the current state we have to apply all previous events. It avoids the replication of data that the snapshot model suffers.</td>
</tr>
<tr>
<td></td>
<td><strong>Process based</strong>: If we are interested in the evolution of natural processes through time we need to keep the integrity of topological relationships maintained with versioning techniques [33] and Voronoi decompositions [34] as in an event based model all changes are treated equally, and there is no notion of event that lasts in time.</td>
</tr>
<tr>
<td><strong>Object</strong></td>
<td><strong>Amendment vector</strong> [31]: Objects (roads, lakes, …) are represented with vectors (lines, boundary lines) and a system stores changes over time.</td>
</tr>
<tr>
<td></td>
<td><strong>Space-time composite</strong> [35]: Space is divided into regions and the system stores historical information related to each area. The region remains unaltered maintaining its spatial topology over time.</td>
</tr>
<tr>
<td><strong>Combined</strong></td>
<td><strong>Triad</strong> [30]: Space, time and feature are treated separately to offer queries that manipulate and relate each component.</td>
</tr>
<tr>
<td></td>
<td><strong>Three domains</strong> [36]: It uses temporal, spatial and semantic domain as organizational bases. The semantic domain serves to store data of events, processes like forest burns.</td>
</tr>
</tbody>
</table>

The presented data models are suitable for computers to store and operate with the space-time dataset. However, we, as humans want to interact with these datasets, making it necessary to develop visualizations and interaction techniques that are more suitable for us.

### 3.2 Visualization techniques

All these models can be approached with different visualization techniques. One of these approaches is using 2D representations derived from classic spatial visualization using maps [37]. **Temporal animation** shows the variation of data over time by changing time layers with a time slider. This technique uses humans’ short term memory but fails to visualize long term data. On the other side, the **small multiples** (Figure 1) technique presents several time layers in space at the same time. Other techniques like **linked windows** approach the problem letting the user interact with the data using different views. Each of the views can present a different representation that aids the visualization of a component or group of them. Another approach is to use 3D representations. An example in this group is the **Space-Time-Cube visualization** (Figure 2), based in Hägerstrand’s model [38] where spatial dimension is represented as the base graphic in a two dimensional plane and time is presented along the vertical axis. Using this representation, a space-time path shows how an object moves in space along time. A vertical (along the time axis) path represents no spatial movement in a period, but the available third dimension in the visualization can be used to represent non-spatial data as well.
Some techniques use the third dimension to position 3D glyphs in the vertical axis. An example of this approach is **Wakame** [39] (*Figure 3*), which describes how to visualize a data-set of multi-dimensional data relatively static in the spatial domain like measures from sensor data in different locations. The visualization uses the axis that goes from the location attached to the data and perpendicular to the surface to represent time. Each of the variables can be represented as a vertex in a radar chart. Joining each of these radar chart layers in time forms a 3D version of **2D data vases** [40]. Another example using the third axis is the **great wall technique** [41] (*Figure 4*). Great wall evolves Wakame and tries to solve its problematic of comparing variables between the different regions as they are presented discrete in space. Its solution consists in joining the same variable in space building a plane that resembles a wall along the time axis. However its solution comes at a price, Wakame can be visualized with few interaction from the user, but in the great wall there is the limitation of representing the connection of one of the variables at a time, so users have to decide which variable they want to visualize.

### 3.3 Space-time visualization in pervasive games

Different of the mentioned models and visualizations have been used in pervasive games. Following, we present some of the methods already used and the problematic that emerged from each one.

*Can you see me now?* [42] is a game in which some participants playing in a virtual representation of the real world had to avoid being caught by some participants running in the real world. Data were recorded using the **snapshot model** and it could be visualized afterwards by replaying the whole state of the game. One of the problems detected with this visualization was the difficulty of getting overall visualization of the consumed space. Players could see the
current state of the game with a dynamic map showing each player’s position. Besides the recorded spatial data, the system provided statistical analysis of players’ movements and logs of players’ text messages. The monitoring system used different interfaces to provide all these different data sources using linked views. Despite it, at the end, the plurality of interfaces, as well as the expert oriented and detailed information, resulted to be problematic and the lack of an overall overview of the evolution of the game-state made it difficult to understand punctual game-states. Game-masters observed that the punctual understanding was leveraged by players’ annotations.

In CatchBob! [43], three participants have to collaborate in order to capture a virtual object by surrounding it in a triangular formation. The system logged position changes, annotations made by users to communicate with each other and connection losses. At the end of the staging, in order to evaluate the game, players had to explain their intentions and thoughts while watching replays of their movements along 2D representation of the paths (Figure 5) they followed. As with Can you see me now?, the paths themselves, or moreover, the non-contextualized information was not always enough to understand their intentions. In this case, logs of the annotations that players used to communicate with each other provided contextualization to game-masters.

![Figure 5: Visualization of space time with paths in CatchBob!](image)

PAC-LAN [44] is a variation of the classic Pac-Man played in the real world. A participant playing as Pac-Man has to escape from the ghost players and consume as many pills as possible. Such pills were implemented with Radio Frequency Identification (RFID) tags in the real world that could be scanned using each participant’s mobile phone. Mobile phone GPSs ent information about location that was represented using a Space-Time-Cube (STC) to analyse player behaviour. As a result of the study, researchers concluded that STC path aggregation with qualitative data of players’ actions helped to understand player behaviour.
Finally, Momentum’s game-mastering system used a web application composed of different views. Different systems showed data from game logs, players’ positions, video from surveillance equipment and information of game elements. The combination of these sources was not exactly following the linked windows view approach since some of the systems used proprietary interfaces that could not be properly modified or integrated into the web application. Game-masters expressed their concern about the problematic of obtaining a complete global overview of the game-state causing that even though lots of data were being gathered from players, sources were not combined in a useful visualization.

3.4 Summary

Different approaches arise when dealing with interactions with the space-time domain. Initial visualizations used textual representations and queries to interact with this dataset. Later improvement in computer graphics allowed the appearance of 2D and later 3D representations that are more adequate for humans as compared to previous textual representation.

Pervasive games have used different visualization techniques to facilitate the game-mastering task. In order to provide effective game-mastering, visualizations must facilitate gaining an understanding of the game-state that ranges from individual to more global through a combination of human-entered and automatically generated data.
Chapter 4
Distributed system communication

To finish the literature review, we present an overview of distributed system communication through web services architectures and real-time communication techniques. A distributed system is a collection of software running on different computers of a network that interact with each other through a messaging mechanism in order to complete a common task. We present web services to approach the messaging mechanism and we introduce two of the main architecture styles in Service Oriented Architectures (SOA): Simple Object Access Protocol (SOAP) and Representational State Transfer (REST). Finally, game-mastering interfaces should show the current game-state; therefore, we also introduce real-time network communication and explain some technologies that overcome the problematic of using the aforementioned web service architectures.

4.1 Web services
A web service as defined by the World Wide Web Consortium (W3C) is “a software system designed to support interoperable machine-to-machine interaction over a network”. Champion et al. [45] and Kreger [46] described the architecture of web services (Figure 7) and separated the different actors of the system in service providers, which manage a set of services and their description. Service providers publish their services’ descriptions to service registries, which facilitate the discovery of web-services by service requestors, the consumers of the services. Service requestors in turn, communicate with service brokers to find the appropriate service in the service registries.

Figure 7: Service Oriented Architecture elements and relationships

The main purpose of the SOA paradigm was to create reusable components that provided a set of functionalities enclosed in a service that could be used by different systems. Two main architecture styles have predominated [47] to implement web services’ communication protocols: Simple Object Access Protocol (SOAP), web services which offer a set of arbitrary operations with a heavily formal contract, and the recent approach of Representational State Transfer (REST) APIs which use advantages of the Hypertext Transfer Protocol (HTTP) and its verbs (GET, POST, PUT, DELETE) to define the interaction with resources of the web service.
4.1.1 SOAP

SOAP defines an XML-based mechanism to exchange structured and typed information between systems of a decentralized and distributed environment with the following components [48]:

- **SOAP messaging framework**: Framework to express the content of a message; who should receive it and the level of priority.
- **SOAP encoding rules**: Define a serialization mechanism to represent application-specific data types.
- **SOAP RPC representation**: Convention to represent remote procedure calls (RPC) and responses.

The SOAP messaging framework at the same time defines as its elements:

- **SOAP message**: Basic unit of communication between SOAP nodes.
- **SOAP node**: Each unit in charge of processing SOAP messages and applying the necessary logic to transmit, receive or process a message.
- **SOAP sender**: Node that sends a message.
- **SOAP receiver**: Node that receives the message.

With these, the SOAP message encodes a description of its destination and who sent it. It does not define any response itself, but it lets request/response mechanisms to be implemented by the consumer, if necessary; as the main design goals of SOAP are simplicity and extensibility.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

Figure 8: SOAP message expressed in XML

4.1.2 REST

REST architecture was introduced by Roy Fielding [49] and was used to design the specification of the HTTP protocol. Its constraints try to minimize latency and network communication while maximizing independence and scalability of components’ interactions. To achieve his goals, Roy defined the following architectonic principles:

- **Client-Server**: Separating concerns of the user interface from data storage and logic increases portability due to being able to implement the user interface in different platforms. It also facilitates scalability thanks to the simplification of server components.

- **Stateless communication between systems**: Requests made by any of the system components must not rely in having a particular context stored on the receiver and every message has to contain all the information to be processed. This improves reliability because it aids recovering from partial failures (every request contains all information needed to be
responded) and scalability since the server does not have to manage context and can free
resources easily.

- **Cache**: With cache constraints, clients can reuse responses if explicitly indicated by the
server and reduce interactions. If it is known that a particular resource from a response has
not or will not vary the server can indicate it to avoid interactions between components to
improves efficiency, scalability and overall performance.

- **Uniform interface**: Applying the principle of generality to the interface of the system we
simplify the architecture since an overall way to interact with it is already defined by the
REST architecture. It simplifies design decisions for service providers, and it facilitates
consumers knowing how to interact with them.

Compared to SOAP where the emphasis is in exposing procedures that provide certain
functionality, REST focuses in resources and how these are exposed for consumption. It
specifies two types of elements: collections and resources, where collection is a group of
resources. Table 2 shows how HTTP verbs are used in REST as a common interface to interact
with the resources and collections:

<table>
<thead>
<tr>
<th>GET</th>
<th>POST</th>
<th>PUT</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of the elements in the collection or information of the element specified.</td>
<td>Create a new collection or resource.</td>
<td>Replace the original value of the collection or resource.</td>
<td>Delete the collection or resource.</td>
</tr>
<tr>
<td>Read</td>
<td>Read-Write</td>
<td>Write</td>
<td>Suppress</td>
</tr>
</tbody>
</table>

Even though REST does not state anything about resource naming, proper REST
implementations typically specify resources to be accessed through *nice* Uniform Resource Identifiers (URIs) specified with Uniform Resource Locators (URLs):

- **Collection**: http://domain.com/collection, where collection is the type of resource
- **Resource**: http://domain.com/collection/element where element is the identifier of the element of the collection that one wants to access

*Nice URIs* refers to the fact that they are **human readable, noun driven** (remember that focus of its API is on resources rather than functionalities) and **describe objects’ structure**. There is no uniform declaration for services that are not related with the previous manipulations, relying on the particular implementation of each API. The inexistence of a language to formally describe the service interface makes of vital importance having a decent API documentation to aid consumers know the data models used to describe the exchanged data.

Exception handling uses HTTP’s standard code status. In **Table 3**, we can see how the errors made by the client use 4xx codes, while server problems are communicated using 5xx. Correct consumption also uses HTTP codes to communicate information about the data exchanged.

<table>
<thead>
<tr>
<th>Correct response</th>
<th>Client error</th>
<th>Server error</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 Ok</td>
<td>400 Bad Request</td>
<td>500 Internal Server Error</td>
</tr>
<tr>
<td>201 Created</td>
<td>401 Unauthorized</td>
<td>504 Gateway Timeout</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>403 Forbidden</td>
<td></td>
</tr>
<tr>
<td>304 Not Modified</td>
<td>404 Not Found</td>
<td>405 Method Not Allowed</td>
</tr>
<tr>
<td></td>
<td>408 Request Timeout</td>
<td></td>
</tr>
</tbody>
</table>
4.1.3 Data transfer formats
Different formats have been used to exchange data when using web services [50] being the main ones the extensible mark-up language (XML) [51] which is used as the main exchange data format for SOAP. XML is criticized for its verbosity and overhead to parse [50], which propitiated the appearance of JavaScript Object Notation (JSON) [52] as an alternative to exchange data between web applications and servers. The motivation behind the creation of this format was to provide a more natural language for web applications compared to XML. This was accomplished using a subset of the JavaScript language, the main language in web-browsers to execute logic to describe the format.

4.2 Real-time communication
The mentioned architectures sit on top of the HTTP protocol, which suffers from not being able to provide communication started by the server and directed towards its client, also called push communication, as HTTP is half-duplex. Thus, meaning that transmission of data can just be done in one direction at a time. In SOAP and REST, the client initiates the communication requesting certain information to the server. The problem with this process is that in order to get the current state of the system, which is typically what is being represented in the web-gamemastering interfaces, the client has to make a petition to the server to get the resources or collections every time the interface requires them. These petitions can be done through Asynchronous JavaScript and XML (AJAX) at a certain rate following a polling strategy (Figure 9).

Polling is not suitable for scalability purposes: if we have a large number of users, the server might end up processing potentially unnecessary petitions as clients do not know when updates in the system are made, and, therefore, clients do not know when a new request is strictly necessary. The other problematic related with this strategy is the uncontrolled delay associated with the difference of time between new data is available in the server and the time that it is communicated to the client. Depending on the rate of polling and the time of the change, the delay will vary.

The way to solve the lack of information in the client related to changes in the server’s data is moving the responsibility of communicating changes to the entity with that information: the server. Therefore, the server communicates clients when there are new data, following a push
strategy. There are different strategies to implement push strategy, WebSocket API defines a simple protocol over TCP to allow the establishment of full-duplex (bidirectional) communications. Clients can open connections to servers that can be used by this last to communicate information towards the client without prior request, leveraging network bandwidth.

![Push strategy diagram](image)

Figure 10: Push strategy

Another technique to push content is long-polling using HTTP requests as used, for example, by Amazon Simple Queue Service\(^7\). The technique only requires the server to keep the HTTP request open and send the response once there are new data in the system instead of answering right away when the petition is done. The problem with this approach is that the timespan that a petition is open might vary between browsers, adding control logic on the client-side. Additionally, in case of trying to minimize the transmitted data, HTTP headers add overhead compared to WebSockets’ 2 bytes requests adding throughput in small transmissions that can potentially affect the performance of the system. A slight variation of long-polling that avoids the overhead of the HTTP headers consists in leaving the connection open and stream new content as it is generated keeping the connection alive. An example of web service using this technique is Twitter’s v1.1 Streaming APIs\(^8\).

### 4.3 Summary

Service Oriented Architectures facilitate network communication between systems. Even though SOAP and XML where the predominant architectures and data interchange format in the beginning, the focus has recently been moved towards REST and JSON which has shown its suitability with the exceptional performance and scalability of HTTP that was designed using a REST architecture.

Acquiring current state of the system can cause clients polling information from the server. This strategy is not suitable for scalability and can be solved using a push strategy: server sends information to clients when something new happens. Implementations of push strategy can be accomplished with WebSockets.

Game-master’s interfaces and in general, the systems of pervasive games interchange data about the elements of the game; thus being REST architecture a better choice as compared to


\(^8\) [https://dev.twitter.com/docs/streaming-apis](https://dev.twitter.com/docs/streaming-apis)
SOAP due to its better suitability for resource based representations. To deal with real-time problems we can use long-polling or WebSockets API to leverage bandwidth problems.
II Methodology

The software development process used for the project has been a variation of the iterative waterfall [53]. The technique includes the same structure as waterfall with phases like analysis, building, testing, review and deployment at a macroscopic level for the extraction of a general architecture of the system in the analysis step; but it applies iterations to improve the systems as the game-master interface system is developed towards the final version.

We start with the analysis of the project introducing the context of the game Codename: Heroes and modelling of the current structure. As part of the analysis, we list the functional needs of the game-master interface based on the game-mastering functionalities and with these we conclude the analysis providing the use-cases to full-fill the requirements. In design we develop a system architecture to implement the use-cases and decide on technologies at an overall level. Finally, the implementation presents the realization of the different subsystems and evaluation offers results of the functioning of the systems.
Chapter 5
Analysis

In this section, we present the analysis of the particular context of the game Codename: Heroes. We start explaining the entities of the game and its attributes as well as the associations between each of the concepts that are going to be used. Secondly, we explain the generic requirements of the game-mastering functions and put them in the context of the game to complete with functionalities that are more specific of Codename: Heroes to extract a list of functional requirements and use-cases that fulfill those requirements. Finding the requirements will provide us a base to fundament the different decisions taken during the development of the architecture in the next chapter as well as identify the different logic entities of our system.

5.1 Codename: Heroes

During 2012, Mobile Life developed a game engine and the game client for a pervasive game that is partly automated and partly game-mastered. The game, called Codename: Heroes, presents gameplay that addresses large scale and long lasting pervasive game by avoiding game-mastering of individual situations [54].

5.1.1 Game

In Codename: Heroes, players have to carry and move virtual messages in the real world in order to gain mana and progress in quests. With the gained mana players can invoke rituals which provide unique abilities.

A typical session of Codename: Heroes would start with each of the players holding their phones and checking an application that shows players the messages they hold as well as the messages that are nearby. For the messages they have in their possession two things can happen: they need to be carried to their destiny or the message is for the player and therefore can be read. Players need to find the location of the destiny using a compass in the application that gives information of direction and distance to the destiny of the selected message. Once they have reached the destiny they can place the virtual messages in the destination boxes, items of the real world that can be accessed scanning a QR-code in the box. By repeatedly carrying messages the player gains mana that was stored in the message. Once the player has enough mana to activate rituals he or she can use mana drainers, a ritual that allows placing a virtual item in a real location to remove mana from players passing nearby. Quests provide a common story for players and make them interact having to go to certain places, activating rituals or moving messages to advance through the story of the stages of the quest.

The main entities in Codename: Heroes are represented in Figure 11.
**Player**: The players in the system. Meaning partly the actual person, but primarily the user in the system and the information stored there:
- **Mana**: Amount of energy that the player has, in three types (one for each kind of team). It is consumed by invoking spells and grabbing messages.
- **Team**: There are three types of teams of players: heart, head and hand. The team has a strong implication in the game by determining how player receives the mana from messages and a softer one since the team of the player should make players role-play in a way that reflects the team believing.
- **Spells**: Special rituals that can modify the state of the game and consume player’s mana.
- **Experience**: Gained through completion of quests and allow getting spells.
- **Carried Messages**: If the carried message has the player as the recipient, this can read it or otherwise is a message that the player is moving towards its recipient to gain mana.
- **Latest known location**: Latitude and longitude indicating player’s position.
- **Quests**: List of quests that a player is currently participating.
- **Fake player**: There are non-player characters (NPC) which are created by the system to carry messages.

**Quest**: It provides framed experiences to players through completion of small tasks. Since the game itself does not have a start or end, these elements help to provide a sense of accomplishment and progression. By completing quests players gain powers, mana and knowledge.

**Message**: Elements of the game that carry information and while carried to its destination, provide energy to players.
- **Recipient**: Who wrote the message
• **Destination**: Where the message has to be delivered, either a player or ritual.
• **Location**: Where is the message: either a player or a ritual.
• **Mana**: Energy carried by the message which can be transferred to the players when the message towards its destination.
• **Content**: Text with a certain information.
• **Decoded**: Messages can be encrypted data. They can just be decrypted by the addressee.
• **Public**: Level of visibility of the message, if private, just the player carrying the message can see it. Otherwise, the message can be detected by other players using their scanner.

**Ritual**: the objects in the game. They can be of different types (some of them):
• **Mana well**: They work as a place for messages to end up. They are virtual and have no visibility in the everyday world.
• **Spells**: rituals activated by players that cause different changes in the other objects. Mana drainer, for example, grabs mana from the objects of the game in a nearby distance.
• **Activity Scanner**: players have one and lets them detect the messages nearby.

### 5.2 Functional requirements

To properly game-master a pervasive game three main functions are needed [1]: ability to monitor the game, influence the current game-state and being able to decide how the game should progress.

#### 5.2.1 Monitoring

An important aspect for being able to make decisions over players’ actions comes from the visualization of the current and previous state of the game. This data are obtained monitoring players’ actions, which in the case of *Codename: Heroes* is using smart phones. Pervasive games are played in the physical world, making time and space the two principal data domains of the obtained data. Nonetheless these are not the only domains covered: from the interaction of players with each other and with the objects of the game we can generate useful social data that expand the understanding of players’ actions.

In Subsection 3.3 when analysing visualization techniques used in pervasive games we have seen that game-masters observed difficulty to interpret data based solely on the current state. A common solution to aid the comprehension was in general using historical logs as well as annotations; therefore the system will present these types of information to game-masters. From all this we can extract the next list of requirements:

- The system shall provide game-masters a map overview with location information about elements of the game that contain location-based information (players, rituals and messages).
- The system shall provide game-masters a timeline with representation of players’ actions.
- The system shall provide game-masters a list with all the items of each element type in the game (players, messages, rituals and quests).
- The system shall provide game-masters a view of the information of each item in the game (player, message, ritual and quest).
- The system shall provide game-masters options for searching and filtering elements in lists.
- The system shall provide game-masters options for searching and filtering elements in the map.
- The system shall provide game-masters means of annotating any element of the game (player, message, ritual or quest).
Tracking system and cooperative game mastering

In an interview with Monitor Celestra’s game-master, he mentioned the use of Post-its around the screen of the shared computers’ screen to keep track of relevant information and changes that a game-master wanted to communicate to the next game-master when they had to change their positions. Therefore, the system can provide active knowledge of other game-masters’ interventions. Thus meaning that whenever a game-master creates an in-game message, the group of game-masters should receive information of that action. The notification system makes game-masters sure of the knowledge they have about the system and it avoids them having to look all the information in the system to reassure that they know the changes introduced by others. Adding the final requirement related to monitoring:

- The system shall provide game masters a list with other game masters’ interventions in the game.

5.2.2 Influence game state

To achieve influence, the easiest way is to directly communicate with players [20]. This communication should not expose the existence of the game-master since its figure should be unnoticeable [20]. In Monitor Celestra [22] there was a different role from the game-master that was presented as phantoms used to directly talk and manipulate what players were doing. This communication with the player can be virtual as well. Examples of virtual communication can be observed in Momentum, where a chat system allowed the players to interact with unusual characters which were played by game masters to try to alter users’ actions. When this technique is used, a main concern arises; keeping track of communication with players and create awareness. If several game-masters take part in a game, all of them have to be aware of the interventions of the other game-masters in order to avoid the creation of inconsistencies that might break the credibility of the game [20]. The other way to communicate with players is through the modification of the elements around the player. These elements can either be real or virtual. During a game of Codename: Heroes, one of the game masters realized that one player was approaching game-master’s headquarter. Soon the game-master realized it was because one of the messages carried by the player was set to be delivered, by error, at his headquarter. To solve the problem without exposing himself or directly moving the destination confusing the player, the game-master tried to set a message near the player with information about an essential event happening close by with the purpose to make her go for it and while far from the headquarter, change the destination of the initial message. Altering a virtual object was the easiest option but it did not succeed, so one of the other game-masters had to dress up as one of the enemies of the player and patrol in front of the headquarter building’s door to discourage her from being around. Both options performed by the game-master tried to keep the consistency of the game. As mentioned, manipulation of virtual objects is easier than altering the real world because it does not require human resources but instead relies on authoring tools.

In Codename: Heroes the communication with the players comes from the possibility of authoring messages. These messages can be written in the name of existing NPCs, which allows game masters to remain unnoticeable.

Therefore, the list of functional requirements related to this function is as follows:

- The system shall provide game masters the possibility of authoring messages.

5.2.3 Authoring

In order to react to player’s freedom of actions, we need to author new elements that can correctly respond to them. Even in the case when players follow what the game-masters expect, the environment could affect existing elements in the game: for instance, construction work in the street where a valuable object of the game was situated and which makes it impossible to access it. Thus, in such changing and unpredictable conditions the system should support authoring [13]. We can, therefore add the next requirement:

- The system shall provide game-masters the possibility of authoring elements of the game (players, messages, quests and rituals).
Several authoring tools exist for pervasive games in general [56, 57, 58, 60]. Most of them cover creation of content before the game starts running because the game has been designed as a whole static experience. Nonetheless, in the case of long-lasting pervasive games, game-masters can use the knowledge about the performance of players to adapt the new content [6].

**Evaluation of game performance**

In a long lasting pervasive game, the creation of new elements should take into account players’ performance to allow them properly enjoy the game. Thus, the system has to provide means to evaluate how a certain element is performing with players.

The creation of new content in *Codename: Heroes* is mainly through quests. These provide players with sets of goals that they have to accomplish in order to advance through the stages of the quest. A number of initial quests exist, but as the game progresses new quests should have to reflect players’ previous performance. The constant creation of content gives game-masters an opportunity to dynamically adapt the creation of quests, not just based on their decisions to steer the history but also to respond to players performance: if a quest was designed to make players do a particular task in a lapse of time, but most of them failed, it could be due to an overestimation of players’ abilities. Adding the last requirement:

- The system shall provide game masters means to visualize player’s performance in quests.

**5.2.4 Use cases**

In the previous point we have extracted the functional requirements based upon the game mastering functionalities. With these we extract a series of use cases to full-fill the functional requirements list.

**Authentication use cases**

These use cases depict the authentication of game-masters as users of the system for security reasons. All the other use cases have Login as pre-requisite.

![Authentication use cases](image)

**Players use cases**

As we can observe in Figure 13, the way to full-fill monitoring requirements of the unitary objects starts with use-cases that expose global views of all the objects of each type. This includes list with the elements (*logical representation*), representation in the map (*spatial representation*) and in the timeline (*social-time representation*). The influencing requirements are full-filled through sending messages to players use case.
Rituals use cases
Rituals have similar use cases to players but they do not generate actions, making rituals have just logic and spatial representations.

Messages use cases
Messages offer similar monitoring use-cases than rituals but they include use-cases for authoring them to full-fill authoring and influencing functionalities.
Quests use cases

Quests full-fill monitoring in a similar way to all the previous type of objects but they include visualization at a more granular level of the sub-quest to provide performance data.

Figure 16: Quests use cases

5.3 Non-functional requirements

Apart from the described functional requirements, the game-master interface for Codename: Heroes has the next non-functional requirements:

- The system should be easy to access.
- The system should provide real-time data of the game-state.
- The system should be secure.
- The system should be easy to modify and adapt to other pervasive games.
- The system should be able to connect to the server that runs the game logic.
- The system should not affect the performance of the server running the game logic.
- The system should be easy to use.
- The system should be scalable.
Chapter 6: Architecture

In this chapter we present a generic architecture for interfaces of game-masters of pervasive games using the requirement specification presented in the previous chapter. After finishing this chapter we will be able to take the last step and decide on particular technologies that are abstracted at this point to obtain an architecture that is not technology dependant.

One of the main goals in the design of the system has been modularity of its components to ease simplicity of the different components. This modularity is accomplished by following web-service architecture and facilitates reusability of the components and the possibility to use already existing systems that full-fill each of the individual tasks. Additionally, we increase scalability by being able to distribute the components to ease the workload.

The requirement specification has led to the identification of four systems whose communication can be seen in Figure 17:

- **Annotation System**: Is in charge of managing the manual observations of the game-masters about the objects of the game.
- **Logger System**: Stores historical data, events produced by game elements interactions and events with game-master activity. These three data types are covered with a generic logging system because all three can be reduced to a system that reads the current game-state and generates some information with it (events, positions or warnings).
- **Game master interface**: System that game-masters use to interact with the game
- **Game engine**: Existing component that manages game logic.

![Figure 17: Systems and relationships of the proposed solution](image)

6.1 Game-master interface

It represents the component of the system that is dependent on the game. Objects of the game are aggregated with specific objects of the game-master interface: the annotations and the events. Apart from the mentioned objects of the game, it contains objects that are not needed to run the game but provide data to aid the game-mastering functions like the game-master events:

**Annotations**: Commentaries written by game masters containing information that might not be provided by the game.
• **Annotation**: Text with the information.
• **Author**: Who wrote the annotation
• **Target**: Object that is being annotated

**Social events**: Contain information of the interaction of the different objects in the game.
• **Message**: content of the interaction.
• **Date**: Time at which the log was done
• **Position**: Location at which the event was generated
• **Involved objects**: Game objects that the message refers.

**Game-master event**: Information of the interventions of the game-masters with the system that need to be communicated to the group of game-masters.
• **Message**: Content of the warning.
• **Date**: Time at which the log was done.
• **Involved objects**: Game objects that the message refers.

**Historic logs of players’ locations**: Log of positions of certain objects of the game.
• **Object**: Object related to the information.
• **Position**: Location of the object.
• **Date**: Time at which the log was done.

We have chosen a web solution to implement the game-master interface to ease cross-platform accessibility. Most of web applications split responsibilities across client-side (front-end) and server-side (back-end). Traditionally, a client asked for a webpage to the server, and a server-side back-end with a scripting language was responsible for generating the HTML page with dynamic content along with some static content. This was to avoid performance loads for the users of the webpage and for security reasons, as logic executed in the server side is not visible from the client-side. This architecture is known as **thin-client** (*Figure 18*) because most of the responsibilities fall over the server-side.

![Figure 18: Thin-client architecture](image)

Later on, the increased performance of personal computers made possible an architectural move towards **thick-client** (*Figure 19*) architectures. As the name hints, this architecture makes its client-side be in charge of the execution of logic and generation of views moving out those responsibilities away from the server-side.
In this project, the game-master interface runs entirely in the client-side. This has been motivated by the fact that the system has to reflect real-time changes as opposed to static content which could be served once by the back-end and avoid logic in the client-side. The data that we have to visualize is quite complex, making usage of template languages in the client-side particularly appropriate to avoid the back-end having to generate the views. Having the view and controller layers’ responsibility in the client-side leaves the model layer in the server-side. As we have designed the system to be distributed, the game engine already offers an independent API, so that we can directly communicate with it from any system. Therefore, it makes sense moving the model layer to the client-side as well, and leave the server-side just to provide the static content needed to run the game-master interface. We could argue that using the server-side would benefit the decrease of CPU load in the client-side when being used with mobile phones. This is not the case as the game-masters are not constrained by a mobile device since they will use the system through standalone computers which offer enough computational power. With the decision of having a thick-client and a web-service architecture for the overall system we have adhered to Service Oriented Front-End Architecture (SOFEA) [61] for the game-master interface. SOFEA has as principles:

- Communication between the web application and the server application should be using web-services.
- Presentation layer is a concern of the client-side.
- Download of the web application, data interchange, and presentation layer must be decoupled. Meaning that the served web application must be static, and any dynamic content has to be generated in the client-side, thus being compatible with browser cache mechanisms and easy to serve using Content Delivery Networks that offer high availability and performance.
6.2 Logger system

This component has to gather information from a certain online source and generate, if needed, new information as means of logs based on it. The logs generated can be directly consumed by other systems through an API. With these requirements, the system has been designed to be generic and with the purpose of offering it as a web service beyond the usage for this project.

Following, is the description of the components of the system and how they relate to each other as seen in Figure 20.

- **Updater**: class that groups a set of information grabbers and the triggers that use that information. It represents the executable unit.
  - **Rate**: Rate at which the updater is required to be executed.
  - **Date**: Last time that the updater was executed.

- **Information grabbers**: contains information of how to obtain the data through communication with other REST APIs.
  - **URL**: Address at which the HTTP request is done
  - **Method**: HTTP request method used
  - **Priority**: Used to establish a priority order when the updater is executed and its information grabbers are executed.

- **Trigger**: Element in charge of the specification of how the results obtained by the information grabbers have to be processed to output the necessary logs.
  - **Evaluator**: Code that is executed getting as input the result from executing the updaters.

- **Namespace**: Represents each of the users of the web service.

- **Log Object**: Represents the objects that are referred in the events of the system, ultimately the particular objects of each user’s system.
  - **External identifier**: Particular representation that each user uses to refer to their objects. It is used to map between the objects of each system with their representation in the Logee system.

Figure 20: Class diagram of the Logger System
- **Event Type**: specific events that each consumer can define.
- **Trigger Error**: contains information related to the execution of a trigger when it has not been executed successfully.
  - **Context**: input that was used to execute the trigger.
  - **Evaluator**: code that was executed.
  - **Date**: when the trigger was executed.
- **Information grabber Error**: Contains information related to the unsuccessful execution of an information grabber to facilitate debugging tasks:
  - **Error**: Information of the produced error.
- **Event**: entity that represents the generated content after processing the data from the information grabbers.
  - **Information**: Text data.
  - **Tag**: Can be used for indexed filtering.
  - **Involved objects**: Objects that the event refers to.
  - **Event type**: Event type that the event refers.
  - **Trigger**: Trigger that generated the event.
  - **Namespace**: Namespace that the event refers to.

In order to have a better understanding of how all these elements interact with each other, *Figure 21* shows the sequence diagram of the main use-case of the system: execution of an updater.
Figure 21: Sequence diagram of the operation Updater::update
6.3 Annotation system

This system shall provide users means to annotate objects. With this simple requirement, the class diagram of the system is as shown in Figure 22.

- **Annotations**: Each of the particular commentaries.
- **Thread**: Group of annotations that refer to the same element.
- **Namespace**: Each of the users of the system.

The difference between the logger and the annotation system is that the logger automatically generates information whilst the annotation system is driven by user’s usage. Here we are separating concerns to help modularity and scalability of the overall system.

6.4 Game engine

Last but not least, this module will be in charge of logic and model representation of the game. In the case of Codename: Heroes, the game uses a Multi User Dungeon (MUD) engine called LambdaMOO (MUD Object Oriented). LambdaMOO is a network-accessible, multi-user, programmable, interactive system used to create low-bandwidth virtual reality text-based adventure games [65]. Thus meaning that the engine does not offer any graphic engine and all game entities are stored as text streams.
Chapter 7
Implementation

In this section we will explain how the different systems have been developed. Starting from the abstract level of the architecture in the previous chapter we decide on current technologies to construct the final artefact.

To have a better understanding of the overall picture of the system, Figure 23 depicts how the different systems communicate with each other through REST APIs that exchange information in JSON.

7.1 Game system

The communication with the game is through its engine running in a LambdaMOO server presented in Subsection 6.4. The possible channel of communication that this engine offers is over TCP and connection with the server has to be maintained in order to maintain the current state. The main problem of the mentioned possibility is that it relies on LambdaMOO limitations, and this includes no native support to manipulate any of the foremost standard formats to exchange formatted data (XML and JSON). A possible solution for this, could have been using Stunt [66], a project that addresses this problematic adding HTTP support, JSON-to-MOO object conversion and a RESTful database browser to the LambdaMOO server. Since several systems: the players’ mobile game-client and the game-master web interface will interact with the game engine it is beneficial to provide a middle layer that offers decoupled communication from the particular technology of the game-engine (Figure 24). With this, we will increase interoperability between all the systems, a desirable property when designing distributed systems [67]. Having a dedicated independent entering point will also make it easier to encapsulate and change the infrastructure of the system without the clients having to change at all. For example, if we needed to scale up the system to address the increased amount of players we could distribute the database (Figure 25) of the game and use a load balancer at the entry point to distribute the petitions and all these changes would remain unnoticed to the game-clients.
Figure 25: Horizontal scaling available with a game API as an entry point to the game engine

By approaching the communication using a stateless web service, we will avoid maintenance of the state through the connection. This is especially noteworthy in the context of pervasive games as players can be interacting with the game using mobile devices (as happens in Codename: Heroes), which might not offer a reliable and constant connection.

7.1.1 REST API with Express

Deciding between using a REST and SOAP architecture depends on the system [47]. As stated in Subsection 4.3, in the context of an API that has to primarily expose information of game objects, we have decided to use REST due to its suitability for resource-oriented [68] APIs, where the main focus is over the exchange and manipulation of the elements of the system as opposed to service-oriented APIs where the main focus is in exposing functionalities of the system.

The API is implemented using Express, a web application framework for NodeJS, a platform to build scalable network applications.

The MOO API will expose the elements of the game as shown in Table 4.

<table>
<thead>
<tr>
<th>Collection</th>
<th>URI URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player</td>
<td><a href="http://domain.com/players">http://domain.com/players</a></td>
</tr>
<tr>
<td>Rituals</td>
<td><a href="http://domain.com/rituals">http://domain.com/rituals</a></td>
</tr>
<tr>
<td>Message</td>
<td><a href="http://domain.com/messages">http://domain.com/messages</a></td>
</tr>
<tr>
<td>Quest</td>
<td><a href="http://domain.com/quests">http://domain.com/quests</a></td>
</tr>
</tbody>
</table>

Table 4: MOO API endpoints

We have used JSON as the data interchange format mainly due to it offering less overhead than XML in the context of REST APIs [50].

API documentation is available9 (Figure 26) through the Apiary.io [69] REST API documentation web service. This web service offers a mock-up server to test the API without having to have it implemented and offers petition validation which makes it easier for API users to test their communications with the API. Even though Codename: Heroes will not be exposing an API for public consumption, it is an on-going research project where different people will develop components that communicate with the game engine through it, thus making it beneficial to be well-documented.

9 http://docs.pervastered.apiary.io/
7.1.2 Real-time API

Getting the state of the game is crucial, but sometimes game-masters have to actively respond to the current player’s actions. This strives the necessity to get real-time data in the interface. The API described before offers current state of the game when requested which seems to apparently solve it. The problem is that the API consumers will have to actively poll against it to know when there are new data to consume. This is not a smart strategy since the amount of usefully transmitted information is not vast. If we have a look at the new information carried in average in each request (Table 5) to the correspondence end point of the API in a play-test session:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (KB)</th>
<th>Difference (KB)</th>
<th>New information (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages</td>
<td>163</td>
<td>0.103</td>
<td>0.063</td>
</tr>
<tr>
<td>Quests</td>
<td>112</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Players</td>
<td>15.3</td>
<td>0.312</td>
<td>2.039</td>
</tr>
<tr>
<td>Rituals</td>
<td>40.2</td>
<td>0.118</td>
<td>0.293</td>
</tr>
</tbody>
</table>

As the API is based on REST, we can mitigate the amount of information sent using HTTP cache mechanisms. HTTP has different cache mechanisms, being of particular interest, ETags (entity-tags). ETag is a unique identifier generated by the server and assigned to each requested resource. The client can then use it when requesting the same resource again and it will be used by the server to know if the client’s version of the resource is the same as the one in the server, therefore knowing if it is necessary to send all the representation of the resource or it can just send back a 304 Not Modified HTTP status to save bandwidth. Even though this would leverage the bandwidth problem, we would still be polling information from the server.

To avoid the polling strategy we use a push strategy: the server communicates whoever is interested when there are changes in the system rather than making clients ask for it. Server streams changes in the state of the game. A way to provide this streaming mechanism is through implementing a strategy based on the publisher-subscriber messaging pattern.
Design pattern
In software development, a design pattern is a compilation of information that describes the structures, relationships and decisions of successful solutions using a notation in a higher level code to abstract their application and make them reusable over different programming languages and even paradigms themselves. They can be presented in several formats [62] but in general those formats cover points like:

- **Name** of the pattern and context.
- **Description of the problem** with an example of specific problem and solution to the specified problem.
- A **general solution** that contains information of the participants of the solution and how they collaborate with each other.
- **Considerations** about the provided solution: consequences (benefits and inconveniences) about the solution.
- **Examples** of situations where the pattern can be applied.
- **Related design patterns** to put the described pattern in context to others, thus which other patterns are being used in the solution or comparison with other patterns that provide a solution to a similar problem.

Publisher-subscriber pattern

- **Context**: We have a distributed system with several systems that are interconnected and produce different types of information. At the same time, each system has interest in receiving just certain types from all the ones that have been sent.
- **Problem**: How can a distributed system send messages to other systems that are interested in those messages without explicitly knowing them?
- **Solution**: Create a communication infrastructure that allows systems communicate their interest in information of other systems. Such interest is expressed through subscription to one of the topics exposed by the producer system. When the producer system has new information it publishes the content in its particular topic and all its subscribers will receive the information. The elements of the solution therefore are:
  - **Publisher**: system that is responsible of publishing messages into any of the available topics.
  - **Subscriber**: the system that is interested in a particular type of information.
  - **Communication infrastructure**: maintains a list of subscribers and is in charge of delivering publishers’ messages to the respective subscribers of its topic.
- **Considerations**:
  - **Benefits**:
    - Decoupling between publisher and subscriber because they do not have to directly communicate to each other.
    - Better scalability than direct client-server architecture using message caching, parallel operations and other techniques due to having control over the communication channel.
  - **Inconveniences**:
    - Decoupling in the channel of communication causes publishers uncertainty about subscribers knowledge and their particular state because those tasks are moved away from publishers responsibilities.
    - Increased complexity in the communication channel to maintain the subscription mechanism.
- **Specific example of usage**: We can actively access a webpage to see if new content has been added but if we are interested in more than one page we will have to tediously repeat that process for each page. One alternative is through RSS feed readers, subscribing interest
for the RSS feed of each page. Then, whenever there is a change in those pages it will appear in the feed reader.

- **Related design patterns:**
  - Observer: improves decoupling between systems dependencies.
  - Message broker: mediator pattern applied to messaging channels. A third object, the message broker implements all communication between two or more systems to decouple them and encapsulate the behaviour.

![Publisher-Subscriber pattern components](Figure 27: Example of publisher-subscriber pattern components)

As we can observe, some of the considerations about the Publisher-Subscriber pattern were already solved using a REST API. We were not directly communicating with methods of the second system, and the decoupling of knowledge between producer and subscriber was already taken into account as one of the design goals of the distributed system architecture is stateless communication and system independence. Instead, the benefit of choosing this messaging pattern is the possibility to express different interests using different topics. This will move the responsibility of distributing the different messages from the consumer directly to the offered API; therefore decreasing the difficulty of adding new consumers as they will not have to implement any distribution mechanism.

**WAMP (WebSocket Application Messaging Protocol)**

The real-time API has been implemented using the **WAMP protocol** (WebSocket Application Messaging Protocol) [70], which is implemented on top of the WebSocket protocol and provides two asynchronous messaging patterns: Remote Procedure Calls and Publisher-Subscriber. It uses JSON as the message serialization format and it offers implementations of server and clients for several languages including Python and JavaScript, which are the languages chosen for the developed systems. The real-time API exposes the channels listed in **Table 6**
Table 6: Real-time API endpoints

<table>
<thead>
<tr>
<th>Resource</th>
<th>Endpoint topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Players</td>
<td>/players</td>
</tr>
<tr>
<td>Messages</td>
<td>/messages</td>
</tr>
<tr>
<td>Quests</td>
<td>/quests</td>
</tr>
<tr>
<td>Rituals</td>
<td>/rituals</td>
</tr>
</tbody>
</table>

Being each of the message changes in the topic the same representation that the objects had in the REST response making it possible to publish interest in different objects at the same time with the same connection and implementation.

The first time the user subscribes a particular interest in a resource, the channel sends data of the current state of the resource and posterior changes are communicated through the WebSocket connection.

7.2 “Pervastered”, game-master interface

Following is the explanation of details about the implementation of the SOFEA architecture and the reasoning behind the design of the different user interface components.

The chosen SOFEA architecture is implemented using one page application where the whole static content of the webpage is downloaded once and then a Model-View-Controller framework in the front-end is responsible to render the page and respond to all the interactions of the user. Communications with the server-side are done using an API and the server is just responsible of the data management.

7.2.1 The Model-View-Controller design pattern

Model-View-Controller (MVC) was presented in 1979 [63] and was designed to solve the problem of how to connect the user’s mental model of an object and their model in computers in a way that would allow easy maintainability of the interface. Following the format specified before we can define this pattern as [64]:

- **Context**: In computer systems with user interaction, user interfaces are used to interact with the systems. There is a mismatch between the logic representation of the data model and how is presented to users in the interface to follow their mental representation model.

- **Problem**: How can the interface be modularized to facilitate modification of its components?

- **Solution**: MVC separates the modelling representation from the presentation and treatment of actions derived from user input into three layers:
  - **Model**: it manages the direct representation of the data as a computer model.
  - **View**: contains information of how data is displayed to correspond to the user’s representation model of the data.
  - **Controller**: translates user interactions with the system into their equivalent changes in the model and/or the view domains.

- **Considerations**:
  
  **Benefits**:
  - Separation in different layers allows developers to be able to focus and specialize in each of the parts: programming of the interface and/or the model logic.
  - This division also allows parallel development. Using stub classes, classes that replace real classes and can be used for testing because they provide consistent results but
without having to implement any of the features that the original class was supposed to be in charge of; the interface can be tested and developed without depending on the implementation of all the logic in the model layer.

- Support for different representation of data due to independency of the model layer from the view layer. This point also allows:
  - Easy modification of model representation: interfaces change faster than the object models themselves due to evolution of technology and change in user preferences. As the model is independent of the view layer, changes in representation just affect the view layer.

_Inconveniences:_
- Increased complexity due to the addition of three layers.
- Even though the model is decoupled from the view layer, the model layer has to take into account the view layer; if each individual change in the model layer is notified directly to the view layer causing it to update, we could be causing unnecessary view updates. Hence, the design of the model layer should take this consideration into account.

- **Specific example of usage:** If we analyse user interfaces to interact with web pages, the web browser itself represents the controller in charge of translating user interactions to changes in the model and view layer. The view layer is represented by the CSS that specifies how the data is presented while the data layer could be seen as the HTML content itself.

- **Related design patterns:**
  - _Observer:_ pattern that solves the concern of synchronization between the model and the view layer.

_Model-View-Controller pattern with EmberJS_

When implementing the MVC pattern in JavaScript there are several MVC frameworks to choose[^10]. For the purpose of the project we have chosen EmberJS. This decision is due to it offering:

- **Template language** to avoid having to code bindings between the data model and its representation.
- **Auto updating templates or 2-way bindings,** the framework takes responsibility to update any element that refers to a model that has changed. This is convenient since we will be representing the real-time status of the system and will mitigate the responsibility to keep data up-to-date.
- **Computed properties,** which makes it easier to enhance the original data model with properties specific of the game master interface.
- Their data model offers an **adapter for REST APIs** that directly maps the interaction with the data model to calls to the API.

The core concepts of EmberJS cover some specific terminology of MVC but also from their particular approach [^71]:

- **Router:** Is the responsible for managing the application state and it is in charge of converting URL to each state.
- **Model:** Is the object that stores the persistent state of the system. It represents the objects of the system as Ember.js objects and therefore they have to be bound with the particular internal representation.
- **Template:** Describes the user interface of the application. It is written using the template language offered by Handlebars.

- **View**: Responsible of converting user events in the interface to set of actions in the system by communicating with the controller and the model.

- **Controller**: Encapsulates logic behaviour to facilitate communication between the model layer and view layer.

We can observe that their implementation uses the same elements of the MVC but it adds the figure of the Router and the Templates. Being the Router a consequence of the notion of state that is encountered in user interfaces. Since REST APIs are based on stateless, independency between current and previous transactions, the management of the state has to be done by the client-side, in EmberJS this is done through the Router. Templates are a layer that manages the representation of the interface using DOM elements. The data-flow with these new elements is as follows [72]:

![Figure 28: data-flow in EmberJS](image)

0. When processing the URL that the user has used to enter the application the router will call to all the needed models and controllers for the current state.

1. Models will communicate through the API Client with the server. With the data from the server they will instantiate the models.

2. Controller loads the needed views, and models flow towards the templates using bindings.

3. The actions of the user bubble until the router and this will change any controllers needed as a result of its actions.

**7.2.2 Model layer**

This layer will contain the logic elements of the system: each of the resources in the game and a representation of the user.

![Figure 29: Model layer class diagram](image)
- **DS.Model**: EmberJS data store model.
- **User**: Contains information of the game master and authentication data.
- **Resources**:
  - **BaseModel**: Common data of the resources in the system.
  - **Player/Message/Ritual/Quest**: Each one of the resources with specific data.

### 7.2.3 Controller layer

We essentially have two main groups of controllers depending on the cardinality that they represent:

- **Ember.ArrayController**: Represents logic encapsulation related to a collection of Models.
  - **BaseController**: Controls logic of the overlays and the side modal.
    - **GeneralController**: Logic for mixed collections.
    - **CollectionController**: Common logic behaviour related with collection filtering, searching and management of one type of element.

- **Ember.ObjectController**: Represents logic encapsulation related to an individual Model.
  - **SignIn/SignOutController**: Authentication logic.
  - **ResourceController**: Common logic for individual objects.

![Controller layer class diagram](image)

**Figure 30**: Controller layer class diagram

### 7.2.4 View layer

This is the most complex layer containing each of the different visual elements and its behaviour.

- **Ember.View**:
  - **ResourceSummaryView**: Summarized representation of an individual resource. Used as the representation when appearing in lists.
  - **MapView**: Map visualization.
Chapter 7: Implementation

- **GeneralView**: Generic view containing a map used to visualize all the resources of the system at once without a list.
- **ListMapView**: Mixes collections’ visualization in a list with their spatial-temporal representation with a map and timeline.
- **DisqusView**: Annotations view.
- **ResourceView**: View of an individual resource with all its details.
- **ListView**: Visual representation of the resources stored in a collection.

![Figure 31: View layer class diagram](image)

**Templating**

The approach for the interface has taken into account the requirement of monitoring. The way to facilitate monitoring of the system is through a dashboard [73] “a visual display of the most relevant information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance”. The main layout of the interface is as follows:

![Figure 32: Structure of the web application](image)
A bar in the top can be used to display the amount of events and access them. On the left, a navigation bar to change between the content of the views which is shown in the widest component of the interface.

The views of the interface can be classified into two categories:

- **Game object** visualization where information of the objects of the game will be shown.
- **Generic view** with general information not directly related with the visualization of the game objects: general annotations and the login screen.

**Game object visualization**

We will keep consistency in the visualization of the objects, no matter their type, to make it easier learning how to use the interface. To visualize their information two levels of detail are provided:

- **Summary** presented when listing collections of game objects. It will provide an overview of the information related with the object.

![Figure 33: Summary HTML view](image)

The information will be presented using the distribution shown in Figure 33. The iconic properties try to convey information using pictograms that represent the name of the property to free the interface of excessive text. In case of confusion the game master can just place the mouse over the icon and hovering text will show what that icon means.

- **Full view** showing complete details of the element.

![Figure 34: Full view HTML](image)

The view is divided into three components: top bar with fixed information of the object as name, id and space-time. A tab panel with the particular attributes of the element and the interactions that each element can have with the other elements. Finally the annotations for the specific object.
**Quests**

When visualizing the objects of pervasive games we will have to deal with specific types of each game, in the case of *Codename: Heroes*, we encounter staged quests.

Quest structure is based on a group of interconnected stages. The navigation between the stages depends on how players behave and there is a definition of conditions which determines the progression through the stages. They are based upon MOO rooms’ structure with the addition of a condition in each of the rooms that decides where to go next. This can be abstracted as a finite state machine [79], which graphical representation can be done through directed multi-graphs: set of nodes connected by one or more edges with a direction and a certain information about the state in each of the nodes. Therefore, we have chosen this graphical representation in Figure 35 to visualize quests:

![Figure 35: Quest visualization](image)

The arrows have been removed since stages in quests cannot be revisited, or otherwise, their graph representation does not contain cycles, which is similar to the definition of the subset of graphs known as ordered trees. We have used a similar representation to the one used for ordered trees fixing the starting stage as the root node of the tree on the top and conveying the order with the readability direction to remove unnecessary arrows.

As mentioned at the beginning, authoring has to take into account the evolution of the game and to do so it is beneficial having ways to evaluate the performance of the authored elements in the game. In *Codename: Heroes*, these are the Quests, for which it has been enhanced their representations showing information on how players interact with them. Players progress through a quest by visiting some of the stages in the quest and this information is shown in each of the nodes (Figure 36).

![Figure 36: Players in each sub-quest can be seen in the nodes](image)

This allows the game-masters to have a direct overview on how a particular player is progressing in a quest compared to others and also, how does the whole structure of the quest is performing. If a particular branch of the quest has a few players, while the quest was designed to make the majority of the players continue through that branch, is that there is something
wrong in the designer’s assumptions. Showing the number of players in each stage does not properly convey information of the accumulative performance of the players since it just represents the current state of the game. To achieve transmission of the historic performance, the game should have to track the amount of players that have been in each of the stages through the lifespan of the quest. This data could be represented using a Sankey diagram, where the focus is made upon representing the flows within a system which help to identify dominant contributions [80]. In Figure 37 it can be seen how the flow of players could be superimposed to the quest visualization.

Figure 37: Sankey diagram to visualize accumulative flow of players

Space visualization
In the previous game-master interfaces of pervasive games it has been observed how space has been one of the most useful data [1] provided by the system. Therefore in the developed interface, the space visualization occupies the vast majority of the interface.

In the Codename: Heroes game engine the space is aspatial: divided into chunks that have a direct mapping with elements of the real world [58] as stored in a database instead of simulating a virtual world. We need ways to model the representation of objects on the physical world. Smart phones provide such representation through means of GPSdata. In that sense, the Earth is modelled as a flat surface, leaving altitude out of typical three-dimensional latitude, longitude, altitude representation. Position is likely to vary along time, due to people being one of the actors of the system; having to incorporate it to the positioning model obtaining three-dimensional model based on latitude, longitude and time. The main problem with this data model is that latitude and longitude are not a proper representation for humans since we are not used to work with them. If we analyse its visualization we can come up with two main representations: either graphically in maps or textually with the related address.

The main problem of mentioned representations is that the user cannot properly identify the data represented if the level of accuracy is too high. This problematic is naturally addressed in most of the digital maps thanks to zooming. With the textual address representation we should either allow the user to change its precision similar to zooming a map or directly choosing a proper level of accuracy. Even with that, the address does not represent precisely the latitude and longitude data so this lack can make it inappropriate for reliable representation, even though it can still be useful to give a better overview of the position compared to latitude and longitude representation.

Another problematic related with GPSis that we have to be aware of the precision of these systems, so we have to deal with inaccuracies of 15 meters for GPSand 2 to 100 meters for assisted GPS [67] in the visualization.

Map with LeafletJS
There are several options available to work with maps in web browsers, being the main ones: Google Maps, Bing and Open Street Maps. We have chosen Open Street Maps because it offers free of charge usage. While Google Maps might seem free to use, their terms of service states
that users can use their service if their solution is freely and publicly available [81]. Otherwise users have to purchase the enterprise version of the API. In the case of the project, the game-master interface is only available for them.

We have chosen LeafletJS that enhances Open Street Maps with an API that offers interactive objects to aid map visualization. It also allows third-party plugin creation to satisfy particular needs over the basis that it offers. In the following section we cover the plugins used in the solution that are particularly useful for the domain of data in pervasive games.

**Marker clustering**

Visualizing positions in a map might not be a problem if the represented positions are widely distributed but in pervasive games we will have players interacting with each other as well as with the elements of the game. As a result of this, the domain of positions that we are going to deal with is a dense one. To facilitate visualization of dense spaces of markers it can aid the technique of clustering. This technique consists in grouping nearby markers. If the user wants more detail, the markers spread out when clicking the cluster to make it easier to select an individual element.

![Figure 38: Marker clustering technique](image)

**Labelling**

Another problem with dense spatial domains is that if we want to add text to help identify the items we will end up facing a mess of texts. We have decided to minimize text shown in the map at the same time by showing text in labels just when mouse is over an element using a labelling plugin.

**Time visualization**

The other dimension, time is usually stored in systems in formats that suit computer representation rather than human readability. *Codename: Heroes*’ game engine is using the Unix time format: number of seconds that have passed since midnight of 1st January 1970 [54]. It is convenient for computers to use this format but it is not that useful for humans. We are used to work with dates in a semantic representation. In this case, we could use absolute textual representations or relative. In some cases, where we want to give information about the time that has elapsed, giving a relative representation might be more suitable (2 weeks ago) than an absolute representation (2nd May 2012).

In *Codename: Heroes* time is present in each of the elements as follows:

- **Players**: last time that the player was connected.
- **Message**: last time that the information in the message was updated.
- **Ritual**: time at which the ritual was activated and time that the ritual will be active after the initial activation.
Both, players and message timestamps make sense to be represented absolutely, but in the case of rituals, as what is trying to be conveyed is time left until it becomes disabled, it is more appropriate to use relative time representation.

**Timeline with CHAP**

We have chosen to use a timeline to interact with time and represent objects over time. The main requirements to select the timeline were:

- Dynamic item loading.
- Support to interactively change the time scale to allow overview and precise detail of the distribution of events in time.
- Support for time range definition.

We have chosen CHAP [74] after comparing different available solutions [75, 76, 77]. It provides dynamic loading of events, smoothly support to up to 10000 events and scale zooming. It has been enhanced with time range definition as it was not an original feature.

The first iterations included the definition of a single time range and iterated until all the interactions were satisfactory for the game-master. The initial operations with the time range allowed its creation clicking in the timeline and deletion by dragging out of the timeline any of its steps. A result of these iterations was the manipulation of the whole time range. Initially to change the time of each step of the range you could drag and drop it to the new time stamp. In play-testing sessions it was seen that the game-master wanted to check ranges of a fixed amount of time at different times. To accomplish the interaction the game-master had to move individually each of the steps while keeping the fixed amount of time which resulted a bit tedious. To help this manipulation, placing the mouse over the time range will highlight it and then it can be drag and dropped to the desire new time stamp while keeping its length as shown in Figure 40.
Social information visualization

In order to thoroughly understand users’ interactions with the system we need to know what they do. This is mainly done by logging each interaction as a description of the action done. In *Momentum* there were ‘informants’ in the player group to provide a description of what they were doing [6] and also there was a diary where players could write their thoughts about their actions. Game-masters observed in this case that even if thoughts were useful, allowing freedom to each player to write down with their own way made it difficult to understand this information. To avoid the representation problem, in *Interference* [78], the game-mastering system automatically logged game events to try to limit the representation of interactions making them more accessible to game-masters. The problem observed was that some of the logged events were of low-level detail: player has connected, player has moved from A to B, periodic information in most of the cases. *Mixing interaction information with periodic system status made game-masters more anxious* when some system information was not shown in time. Thus periodic information and information related to system’s status should be avoided from the game-master interface.

Finally, events happen in a particular time, which contextualize them. Instead of showing their associated date textually we will use a timeline to represent them. This will help to convey a better understanding of how social events relate to each other by having a visual representation of their distribution in time as shown in *Figure 41*.

![Figure 41: Events in timeline](image)

Connecting time, space and social events with TardisJS

There are no plugins to connect map and timeline for CHAP, therefore we have developed TardisJS to connect them and provide *linked windows* visualization. The connection needed comes from the need to offer these interactions:

- Time ranges are defined in the timeline, then positioning of the defined time lapse is loaded into the map as paths and heat map.
- When clicking an event in the timeline, the game objects represented in the map that appear in the event are highlighted, showing their ids and names.
- Just the results from the search bar are shown in the map and the timeline, being tardisJS the responsible of connecting results of ids into the related objects in map (markers that directly represent that result) and events (objects mentioned in the event per se).
We also use the connection of space and time to show two types of representations in the map.

*Individual space-time representation: paths*

Information of positions obtained over time of a particular player are represented by means of positions connected in order of acquisition. Each of the positions shows information of the time at which that position was gathered and the object that refers to.

To convey *notion of time in space*, the *time range is mapped into each of the paths using circles* identified with the number of the time range. In *Figure 44* we see that the start of the time range in 16th of May is for one of the paths at the middle top of the map marked with a green circle and a white zero, while the other path has it on the top left marked with a pink circle and a white zero. For the pink path the other time range shows at the middle-right of the map. And addition to the initial representation uses *animated arrows to help understanding the movement* within the segments of the path. The animation of the movement in the path is quite expensive and therefore it is just triggered when placing the mouse over the path.
Testing the interaction brought the addition of the possibility to define sub intervals for the range. Clicking any of the steps in the range will create a new step (Figure 45) that will show up in the map as well. It can be manipulated in the exact way as the other steps: drag and drop to change the time stamp and if dragged out of the timeline it is deleted.
Collective space-time representation: heat-maps

Paths are suitable to understand each player’s activity but when game-masters try to get an overall understanding of player behaviour in the space, a heat-map is a better representation. Heat-maps remove the temporality provided by paths and show an aggregation of all the positions using color to indicate the amount of positions in each area.

Heat-maps also help the visualization of paths as one of the main problem with paths is the understanding of dense positioning. For example if we have a look at Figure 47.
We can end up spotting that the bottom left corner is the area most visited since we see the vertex with the highest degree (in terms of graph terminology). It is quite difficult to understand the whole distribution of positions at a glance. Figure 48 shows the version with heat-maps enhancing the paths.

Overlapped positions are enhanced making it easier to understand the real structure of the path.

**Peuquet in TardisJS**

The visualization tool has been designed to provide answers to the 3 questions that Peuquet stated were needed when interacting with time and space domain and that are presented in Subsection 3.1. Following is the description of how they can be answered using the tool:

- **When + where → what:** detect objects that where at a particular position and time
  1. Select a time range in the timeline that includes the time that you want to query.
  2. Select in the map the area that you are interested.
  3. In the map there are paths showing the objects that pass through or were in the area at the selected time.

- **When + what → where:** detect objects’ location at a particular time
  1. Select a time range in the timeline that includes the time that you want to query.
  2. Search the object using the search bar in the list or scroll until you find the object that you are looking.
  3. Click the object to focus it in the map view and check the drawn path to see its position.

- **Where + what → when:** detect the time that an object or group of objects were at a particular location
  1. Select the area in the map view.
  2. Select the object in the objects list. Introducing the name of the object will hide all the other objects from the map and the timeline.
  3. Select a wide time range in the timeline to see if a path passes through the area.
  4. Place the mouse over the point of the part that is in the location that you are interested in to get the time.
7.3 “Logee”, logger system

Different current operating web services offer similar features to those described for the logger system. We have approached the different existing options looking at web services that covered some of the features of the system as it can be seen in Table 7

Table 7: Requirement compliance of different existing web-services

<table>
<thead>
<tr>
<th>Web service</th>
<th>Gather data from other APIs</th>
<th>Create logs based on the data gathered</th>
<th>Log storage</th>
<th>Log retrieval from other systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFTTT</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Amazon CloudWatch, Loggly, Circonus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Herokuapp, Amazon EC2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

IFTTT (If this then that) [82] was created as a response to facilitate communication between existing APIs. Their system is structured in *recipes*, which are composed by three elements: the *channels*, a list of 62 APIs from where to gather the data, *triggers* that are activated when a certain condition about the data gathered is satisfied and *actions* executed by satisfied triggers that communicate to a subset of available channels. As we can see the structure is similar to the system described. It differs in the fact that IFTTT does not provide custom definition of the source (channels in their terminology) and that trigger’s results are not consumed by the web service itself but directly communicate with other APIs. Although we could solve the problem of event logging with communication with file storage APIs like Box or Dropbox, we would still have the problem of communicating with the internal API of the game engine.

Another group of web services that could be used is the ones which allow *system monitoring*. These web services offer tools to log and visualize system parameters that range from network statistics to CPU load and other measurable parameters both from a server-side and client-side perspective. We are not interested in these measures but the ones consumed from an external API. *Amazon CloudWatch Metrics* offers monitoring of custom metrics that the user has to actively send via a PUT HTTP request to their API. A similar option but with support for JSON objects is offered by *Loggly* and *Circonus*. All the mentioned web services allow the creation of alarms when a certain function is satisfied by the logged metrics. Concerning the retrieval of the generated logs, *Loggly* offers a REST API to consume logs and do queries to filter depending on user’s needs. The problem with both web services is that we still have to generate the logs that these web services will consume since the game engine does not create them. Even though we could add a system in the game engine to generate the logs from where to feed the mentioned web services, we would still have the problem that mentioned web services just allow visualizations provided by the service itself and cannot be consumed by an external client. We can see them as one way communication systems in terms of machine interoperability (there is a two way communication if user visualizes those logs).

Finally, the last group that has been considered are web services that offer web application hosting. The purpose is to find a web service that offers web application creation and built-in access to database elements through an API. With *Herokuapp* we could build a web application with custom code to fetch data and use an already existing web framework like Restify or ExpressJS to create the API. This would offer a solution to the problem but we would end up with a solution that would need to be replicated by everyone interested in the system. Therefore we have decided to implement the logger system as an independent web service.
7.3.1 Front-end

The front-end uses the predetermined admin interface offered with Django to allow the creation and edition of most of the elements in the system. Additionally, a customized view (Figure 49) facilitates the task of trigger coding and debugging. Triggers in the system are scripts in JavaScript that receive data gathered from the Information grabbers and generate events as outputs.

Editor with Code-mirror

The code editor of the triggers uses Code-mirror which supports JavaScript syntax highlighting and connection with JShint to offer detection of compiling errors and warnings for JavaScript. This view lets the user choose from each of the triggers in an updater and also which source they want to use as input for the trigger:

- Customized, to debug particular tests.
- From the result of the execution of the information grabbers.
- To facilitate bug fixing, users can select an option to use the result of the information grabbers that have caused an exception.

All previous input methods can be edited to allow further tests based on existing ones.

![Figure 49: Updater edition HTML view](image)

Once the trigger and the input have been selected, any change in either the code or the input text causes the current trigger to be executed. Results are shown in real-time to facilitate coders understanding the results of their changes. Finally, a box shows the validity of the generated result because even if a trigger is executed without errors and a result is generated, this could not follow the Logee API format (see Subsection 7.3.3 for further details about the API specification). In this case, indications of the cause of the incorrect format will be presented as seen in Figure 50.
7.3.2 Back-end with Django

Django has been chosen to develop the back-end, which offers a high-level Python Web framework for rapid development. It has an object-relational mapper which allows data definition in Python to be bounded to the specified database, being MySQL the option for Logee. It offers a built-in front-end administration panel for management of objects in the system.

7.3.3 API

A REST API allows consuming the events in the system and supports filtering by using the parameters described in Figure 51.

Events created by triggers in the system have to be outputted as JSON objects following the format shown in Figure 52.
As with the MOO API, the Logee API documentation is available in apiary\textsuperscript{11}.

### 7.3.4 Game-master objects support in Logee

Logee has been used to represent:

- Events of type **Social** containing information of:
  - Players joining a Quest
  - Players changing of quest stage while being in a Quest
  - Message moving from one Ritual to another location

- Positions of type **Position** which contain information of positions in time, and are recorded when there is a difference of distance in a range of 200m (enough to avoid problems with positioning precision)

- Actions of type **Game-master action** which contain information about changes that game-masters have done when changing a message location.

All previous actions can be represented with a text base data interchange format like JSON because log event objects can contain a string in the information field.

### 7.4 Disqus, annotation system

We can find different web services that cover the features of the annotation system being examples **IntenseDebate** [83], **Livefyre** [84] and **Disqus** [85]. For this thesis project we have chosen Disqus because the free version already fits the needs of the system. It is worth mentioning that the differences between the mentioned services are minor and mostly economic.

Disqus offers real-time authoring of comments. There is an API over HTTP to access the comments and an embeddable code to use out of the box to provide an interface for message authoring as seen in Figure 53.

![Disqus box](http://docs.logee.apiary.io/)

Figure 53: Disqus box

Their API allows the creation of threads using a unique identifier for each one. In this thesis project we use the object internal id to have annotations for each of the elements in the game. Additionally there is a general thread to discuss things not related to a particular object. Another useful feature is being able to subscribe to any particular object’s comments, facilitating the task of cooperative game mastering.

\textsuperscript{11} http://docs.logee.apiary.io/
Chapter 8
Evaluation

In this section we will study the built solution at two levels: user interaction with the system and system technical performance and limitations.

8.1 Interface

To evaluate the usability of visualization techniques we should cover [86] identification of user goals and verification of accomplishment using the provided technique, usefulness of interaction mechanisms to accomplish each goal, identification of visual representations and finally we need to see how visual representations and interaction techniques relate to each other.

We have chosen task scenario evaluation to create a set of tasks that the game-master will have to face using the game-master interface which define the user’s goals. The tasks have been designed to cover the main requirements of game-mastering pervasive games.

The subject to execute the task is the current game-master of Codename: Heroes. He has been chosen because having performed as a game-master gives him comprehension of what it is needed for this role. Having used the previous MOO interface gives him a wider understanding of what is provided by each interface and which interactions work better.

The evaluation has started by describing the functionalities of the system and how they work. Afterwards, one by one and in the presented order, the tasks have been described. The technique of think-aloud [87] has been used to understand the reason behind his actions and experience while using the solution. This technique consists in asking the user to mention details of what he is looking at, thinking about, doing and feeling during the process of completing the different tasks. The reason behind using this technique is that it has been successfully used in previous pervasive-games as evaluation method of players and game-masters[89, 90, 91]. In those studies it has been pointed out the benefits of think-aloud to discover insights that might have been difficult to acquire through other studies like questionnaires [89] which would not make much sense in this study since we just have one individual.

8.1.1 Monitoring

- **Case description:** Find players that need help (with unread messages or stuck in a quest).

- **Detailed description:** The system has data of A) a player that has been in a quest for 5 days. Activity in the event log shows that he has had activity in the game in these 5 days and that he has been trying to complete the quest stage by going to the place where he is supposed to grab a message.

- **Research questions:**
  - How well path representation and event logs help understanding players’ actions?
  - Does information need to be more centralized and explicit (for instance, showing time since quest has been joined directly in the quest interface) or can be easily extracted from other data sources?

- **Observations:**
  - He looked for information represented in ways that he had already been using: relationship of objects. After watching players for a while (3 minutes) to have an overview of the current state he turned to watch quest structure to have an overview of quest stages. Then he observed that most of the players were in the final stages unless one. He looked further information about that player by checking his current position. Immediately after, he has checked the event timeline followed by an overview of the movement of that player by selecting a wide time range (3 weeks period).
Chapter 8: Evaluation

- **Subject observations:**
  - Even though it was easy to grasp an understanding of the current state, having each type of object showing its own layer in the map view was confusing. Also it was confusing remembering the state of his decisions: if he had been checking a particular quest and afterwards a player related to the quest, it would have been difficult for him to remember why he was checking the player. Keeping track of the navigation history would be appreciated. Time ranges were difficult to manipulate with precision and in general, understanding events in time was a bit difficult. Idea of ordering is easy to grasp but a real comprehension of distribution of events in time: morning, afternoon and night was difficult. Having filtering of types of events or making more obvious the different type of events would help to understand them. When zooming in and out or moving the timeline is difficult to keep track of what someone is looking because all the markers look the same.

- **Conclusions:**
  - Paths showing position in time and events help to get an understanding of players’ actions but they have been consumed in a wide overview and mostly perceived in an atemporal way: there is not much interest in precise detail of time. Instead it is just used to create an understanding of order in which things happened.
  - Position visualization with a map is much easier to understand than events in time with the timeline. Then, if trying to convey information related to time is better for it to be more explicit, while spatial one can rely on being open to discovery by the game-master. Thus increasing visual clues of morning, midday and afternoon in the timeline would help understanding the distribution of events in time.

8.1.2 Authoring

- **Case description:** Design a quest based on historic data of player’s behaviour.

- **Detailed description:** The system has data of A) a sporadic player, B) a teenager that plays in the afternoon, C) a worker that plays sporadically alone during the week and extensively with player D) during the weekend, and player D) that plays on Saturdays.

- **Research questions:**
  - Which kind of data is more meaningful: individual (paths) or collective (heat map)?
  - Which kind of data is more meaningful: space or time?
  - How is this data experienced with the representations used?

- **Observations:**
  - Game-master checks heat map to have an understanding of players’ movement. He then cares about individual representation and decides to check paths. After 3 minutes in which he still visualizes combined paths by using the object filter and grouping by the team of the player. Usage of time is avoided and just used to select the data range for the paths and heat map.

- **Subject observations:**
  - Heat maps certainly helped in understanding overall movement and let him use that information to make a first discrimination of how to position the objects for the quest.
  - Paths have been useful to understand which was the origin of the data of the heat map.
  - Paths have been used with few players to avoid the mess created when visualizing all at the same time.
  - He has not used time because it seemed difficult to gather any information from the system and the system did not allow to get accumulative time data: time ranges let selection of a continues range of time but it is not possible to select for example information of all days at a particular time range.

62
• **Conclusions:**
  o Accumulative visualization with heat-map is preferred over absolute representation and relative ordering helps understanding time but detailed information of time is not used.
  o Space is preferred as it is easier to understand.
  o Paths aid relative ordering of positions in time but there is no much interest in precise absolute timestamp of the positions.

### 8.1.3 Altering current state

- **Case description:** Player A) has to pick a message in order to progress and needs the aid of the game master.

- **Detailed description:** The game-master is asked to make a player pick a message (by changing message position and placing in a position nearby) but the player will be reluctant to pick it making the game master react to it.

- **Research questions:**
  o Does path representation helps in the understanding of players’ movements?
  o Are previous actions needed to understand the current state? How do events and position aid in this understanding?

- **Observations:**
  o After watching player moving he has enabled paths and selected a range that includes current date to keep track of player’s current movement. He has placed the message in a nearby ritual that was in the direction that the player was following. He has passed by without taking it. To continue game master has selected 1 month as the time range and checked from the rituals around which was the most visited, after having player not taking the message again it has finally decided to leave the message in his drop box.

- **Subject observations:**
  o Positioning was the most useful domain to understand player’s movements. Events helped understanding which their purposes were. Having experienced this problem before with just the visualization of the current position, having the paths with previous positions helped to understand which are player’s intentions, ending with an easier decision-making.

- **Conclusions:**
  o Previous information helps to understand the current state and space is the domain preferred to consume it rather than looking the events.
  o Visualization of real-time paths alleviates the game-master from having to remember and pay attention to all current active players.

### 8.1.4 Conclusions

Timeline, as it is, does not provide an effective understanding of data even though it is useful to filter the position information shown in the map. One of the main reasons for this difficulty is that the focus of attention is easily lost in the timeline when zooming or translating. In comparison to a map where there are rich visuals and every area looks different, the background of the timeline is uniform and the events looked quite similar. Therefore the timeline should provide visual context to help keeping track of the focus when the user performs zooming or translation. A possible solution could include using colours and icons for the events and using a pattern for the background of the timeline.

Heat-map is useful to develop a fast understanding of players’ movements, and paths facilitate understanding of each player. The integration with the timeline is used mostly to filter and select a certain amount of positions rather than to know at what time each movement happens.
Historic context of game-master actions in the interface would help remembering what game-masters are doing in the current view.

8.2 System

The system evaluation will cover all the systems described in Chapter 7: the logger system, game system, annotation system and game-master interface. For the setup of the implementation we have decided to separate each of the subsystems used in different machines as if they were managed by their respective provider. The evaluation of the systems is made studying performance metrics from two perspectives: client-side, where we check the response time that will provide insights in responsiveness of the system; and server-side looking at CPU load and RAM consumption to study performance and scalability of the system. In order to simulate simultaneous users we have developed a JavaScript that makes asynchronous HTTP requests based on three parameters:

- **URL**: the endpoint of the API call to test.
- **Interval**: time in milliseconds to wait between HTTP requests.
- **Repetitions**: Number of requests to make.

For example, choosing an interval of a few milliseconds and a high number of repetitions will simulate simultaneous users while an interval higher than the response time of the call will simulate individual usage.

8.2.1 Logger system

The logger is deployed in a Digital Ocean\(^\text{12}\) cloud hosted machine located in Amsterdam with the following hardware specification:

- 1 CPU Intel QEMU, 2 GHz, 4 MB cache
- 512 MB RAM memory
- 20 GB Solid State disk

At a software level, the top processes in the machine are as follows:

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU Load (%)</th>
<th>RAM (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>1</td>
<td>180</td>
</tr>
<tr>
<td>Mysqld</td>
<td>0</td>
<td>79,8</td>
</tr>
<tr>
<td>Nginx</td>
<td>0</td>
<td>7,92</td>
</tr>
</tbody>
</table>

- **Python** is the process running the instance of the Django framework.
- **Mysqld** is a binding that allows Python communicate with the MySQL database.
- **Nginx** is the reverse proxy that proxies the HTTP petitions and follow them to the Django framework.

For the test of this system we wanted to see the impact of two variables:

- The number of events consumed to define the limitations of the events loaded from the timeline and map tools.
- The number of simultaneous users of the web-service to know the amount of simultaneous game-masters that could be using the historic data.

---

\(^\text{12}\) [https://digitalocean.com/](https://digitalocean.com/)
Server-side results

CPU load of top processes based on simultaneous users and events returned.

Figure 54: CPU load of Python process

Figure 55: CPU load of Mysqld process
Chapter 8: Evaluation

Client-side results

Response time for petitions to the logee API.

Analysis

The analysis of the logger event depends in the requirements of the consumer of the web-service. We do a global evaluation of the service posing its current limitations and then analyse the results in the context of the use for the game-master interface.

The results obtained show that the Django Python application is the most cost effective process in terms of CPU load and RAM consumption. In Figure 54 we can observe that between 11 and 47 events in the response increases the CPU load from 11 to 65 % when having 50 simultaneous requests.
users. Looking at the client-side perspective (Figure 57) we see that 47 events in the response already bring the response time to around 5 seconds. Looking with more detail the lower range of number simultaneous requests (Figure 58) we can guarantee a correct functioning with response times of up to 5 seconds for requests to the API with around 50 events per response and serving up to 50 simultaneous users. This causes a 60% CPU load in the current server configuration.

![Response time](image1)

Figure 58: response time for requests to the Logee API zoom in lower bound

![CPU load](image2)

Figure 59: CPU load for Python process zoom in lower bound

To help understanding the limitations we have to put them in context of the use that the game-master interface makes of the service. To get a better understanding of the usage of this service by the game-master interface we gathered information of the number of events generated by players during 10 play-sessions.

<table>
<thead>
<tr>
<th>Time (min)</th>
<th>Events</th>
<th>Distance (Km)</th>
<th>Velocity (Km/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.07 (5.26)</td>
<td>9.29 (4.68)</td>
<td>0.978 (0.55)</td>
<td>2.85 (1.63)</td>
</tr>
</tbody>
</table>
Choosing arbitrarily a play-time of 1 hour per day per player we can estimate this amount of events to be generated:

<table>
<thead>
<tr>
<th>Players</th>
<th>Day</th>
<th>Week</th>
<th>Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>48</td>
<td>336</td>
<td>1440</td>
</tr>
</tbody>
</table>

With numbers in Table 10 we can conclude that for a single game-master, the current web-service can just serve smoothly one player’s daily activity. One way to solve current limitations would be through vertical (increasing power of the machine were the service is deployed) or horizontal scaling (distribute the server load between several machines). Other possibilities to mitigate the repeated consumption of the same events include using cache mechanisms and browsers’ local storage. Applying cache mechanisms would be motivated because historic events cannot be modified, thus needing just to take into account that to cache the resources we would need to uniquely identify the requested time ranges. Arbitrarily we could establish time ranges to be split into each of the weeks or days that it contains making for example that if the user selects in the timeline a range from the 21\textsuperscript{st} of November of 2013 to 26\textsuperscript{th} of December of 2013, split the range into different calls to weeks 47 to 52. Using browsers’ local storage would mean using HTML5’s local storage specification\textsuperscript{13} which offers a key-value store up to 5 MB of persistent data. With it, we could store 50 players’ weekly activity (16800 events) or in case of browsers that allow modifying the maximum amount of allocated memory, much more.

### 8.2.3 Game system

The MOO API along with the game engine are in a server in Ireland using Amazon EC2\textsuperscript{14} cloud instance with the following specification:

- 1 CPU Intel Xeon, 2,27 GHz, 4 MB cache
- 590 MB RAM memory
- 8 GB hard drive

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU load (%)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>0,0</td>
<td>84,5</td>
</tr>
<tr>
<td>Moo</td>
<td>0,1</td>
<td>24,9</td>
</tr>
<tr>
<td>Nginx</td>
<td>0,0</td>
<td>7,7</td>
</tr>
</tbody>
</table>

- **Node** is the REST API to communicate with the MOO server.
- **Moo** is the server running the game logic.
- **Nginx** is the reverse proxy that proxies the HTTP petitions and follow them to the Node application.

\textsuperscript{13} http://www.w3.org/TR/webstorage/#the-localstorage-attribute

\textsuperscript{14} http://aws.amazon.com/ec2/
Chapter 8: Evaluation

The game-state stored in the game-engine at the time of the test contains:

Table 12: Elements and total size of each resource of Codename: Heroes

<table>
<thead>
<tr>
<th>Resource</th>
<th>Size (KB)</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>players</td>
<td>163</td>
<td>38</td>
</tr>
<tr>
<td>messages</td>
<td>112</td>
<td>124</td>
</tr>
<tr>
<td>quests</td>
<td>15,3</td>
<td>3</td>
</tr>
<tr>
<td>rituals</td>
<td>40,2</td>
<td>63</td>
</tr>
</tbody>
</table>

Server-side results

Figure 60: CPU load of Node process

Figure 61: CPU load of Moo process
Client-side results

Figure 62: Response time of MOO API based on resource and simultaneous users

Analysis
The process that causes most of the bottleneck is Moo, the game-engine; that with 500 users requesting information of the current players game-state brings the CPU load to 73.6 % (Figure 61). The Node process (Figure 60) shows that the CPU load of querying each resource is in general directly proportional to its size (Table 12), probably because the only thing that the process does is to act as a proxy with the game engine. Consequently, it just depends in processing the game engine result. On the other side, the Moo process CPU load is not directly dependant on the size of the elements returned. As an example, it is worth remarking that querying information of the quests in the game, which is the resource with fewer elements (3) and size (15.3 KB) as seen in Table 12, ends up causing the worst performance as users scales up compared to querying messages or rituals.

As initially stated in Subsection 7.1, we have designed the MOO API to be a layer on top of the game engine. Thus, we can use memory cache in Nginx to avoid reaching the Node process, nor the Moo process for requests to the same resource endpoint that have been done in a short period. The system would not require that much memory to cache the results as the elements per se do not have sizes over 163 KB.

From the client-side perspective, Figure 62 shows a growing tendency with the number of simultaneous requests but with response times of less than 1 second for 1000 thousand users the responsiveness of the web-service is acceptable.

8.2.4 Annotation system
As we do not own the annotation system used we cannot evaluate technical metrics from the server-side but mention that its JavaScript is of 2,8 KB with response times of 200 to 400 ms and interactions of under 1 second.

8.2.5 Game-master interface
The game-master interface is hosted in the same machine as the game system but could be hosted and replicated in any server as it is just independent static content following standards of SOFEA. It just has to be downloaded once (3,5 MB) and around 4 seconds in having the page ready for the first time and it takes 2,1 seconds to load and render the page from the JavaScript. The memory consumption is of 30-60 MB. Most of the interactions take less than a second, being just the ones related with the timeline the ones sluggish due to the slow response time of the system.
Ill Discussion

This section ends up the thesis talking about the suitability of the different decisions done during the development of the study and remarking the points that are open and that could be addressed in future work.
Chapter 9

Conclusions and future work

This last chapter of the thesis exposes the discussion of the results of the obtained solution.

9.1 Discussion

From the list of research questions in Subsection 1.1, we can say that we have successfully designed in Chapter 6 a service-oriented architecture for a game-master interface that modularizes the different systems needed to full-fill the requirements of the game-mastering function. The implementation in Chapter 7 proposes current technologies and existing web-services to implement each of the systems. For this matter, the best options is following a Service Oriented Architecture using REST APIs to communicate between the systems as it is suitable for scalability. There is not much real difference between using JSON and XML as when compressed they become of similar length. The API could give support to both through previous selection using HTTP headers to let each consumer choose what they want.

Following a service oriented architecture adds a difficulty layer to keep all object representation in the same format as there is certain coupling with the one of the service, thus being of vital importance being able to either allow the user define a custom format for their objects or define a converter offered by the service to facilitate reusability when interacting with that service through other systems. In this thesis project, the logger initially created events that were consumed by other systems and had information of the game objects using the specific format defined by the logger system. After seeing that the conversion of the format of game objects in a log event to their game engine’s format would need to be replicated in every other system to get consistency, we changed the log events to allow user definition of their own serialization method. This same concern arises over the usage of JSON and XML: we could decide for JSON as we had control of all the systems but in a more open service, it would benefit the users having multiple data interchange formats to interact with the service without having to create format dependency. A final consideration about the creation of web-services for pervasive games is that cache mechanisms can easily benefit scalability of the systems as discussed in Subsection 8.2 and therefore we want to emphasize the importance of resource naming and the design of REST APIs that use it. An example of bad design for cache usage was the initial consumption of time ranges made by the logger system as the parameters of the query were enclosed in the body of the request. The request was produced against a generic endpoint used for all the queries and therefore there was no unique identification of the requests to facilitate caching.

On the other hand, by detecting concerns and dividing tasks into services we are able to add reusability to the architecture as initially intended. In the thesis, we use Disqus as annotation system and have developed sub-systems like the logging system that are reusable in other contexts and that can be changed if a better service offering its features appears. Applying SOFEA principles and moving the concerns of the presentation layer entirely to the front-end allows more reusable interfaces as they do not create dependency or conflict with the server-side stack of technologies that can differ depending on the project and game engine. While the client-side stack is limited to JavaScript. One example of this is that during the development of the thesis there was a replication of the game engine to suit the needs of another student. As there was no server-side logic it was easy to setup a copy of the interface to talk with it with the only modification of changing the address of the MOO and logging API that the interface was using.
We have also developed an **interactive visualization tool for space, time and social domains** based on **linked windows view** using a map, timeline and list to represent objects explained in Subsection 7.6. Even though, the first intention was to increase focus in exploration of the time domain, this has not been accomplished. Instead, the timeline has been mostly used to manipulate the date ranges loaded into the map, being successful as a manipulation tool but not particularly proficient at visualizing temporal data. This might be due, in part, to the important amount of space of the map in the interface as compared to the timeline but also to the fact that interaction with maps is quite usual and well-developed while the usage of interactive timelines is rare, making the tactics to explore the combined domains to be almost new to the user.

## 9.2 Future work

The scope of the thesis did not include the complete development of the sub-systems beyond offering the basic functionalities needed from them in the game-mastering interface. To this matter, **Logee can be developed further** to be used as a web-service for other projects as the requirements that this covers are not full-filled by current web-services.

The evaluation of the interface has been done with the game-master of **Codename: Heroes** and using tasks in a limited environment. It would be interesting to **evaluate the usage of the visualization tool in a real long-lasting staging** of the game to get more insights and let the game-master explore the possibilities of the timeline to discover new ways to interact with it. Continuing with the visualization tool, an improvement to be considered would be changing Open Street Maps for a solution with 3D visualization like Google maps and provide a visualization using **Space Time Cube but adding the timeline** and the time range definition developed in this project to improve the manipulation of time.

Extending the area to other fields of human-computer-interaction the problem of visualization and interaction with space-time domains goes beyond the usage in pervasive games and as such it would be interesting to **study and develop further the proposed map with timeline and time-range manipulation in detail** and detached from the constraints of pervasive games. Applications of such study can aid the visualization of data acquired from “wearable technology” that contains space and time domains as well. Furthermore, with services like **Moves**\(^\text{15}\), a mobile application that records movement in space and offers an API to integrate with other services will facilitate the appearance of more complex interactions of data with the spatial and time domains than just path manipulation.

\(^{15}\) [http://www.moves-app.com/](http://www.moves-app.com/)
IV References
References


5. Steinmetz, Ing Ralf, and Dipl-Inform Viktor Wendel. "Game Mastering in collaborative Multiplayer Serious Games."


Benford, S.; Fraser, M.; Reynard, G.; Koleva, B. And Drozd, A. "Staging and evaluating public performances as an approach to CVE research"

"The underwhelming effects of location-awareness of others on collaboration in a pervasive game"

"3D space-time visualization of player behaviour in pervasive location-based games"


Pautasso, C.; “Restful web services vs. “big” web services: making the right architectural decision”

http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ Last accessed February 2013


Nevelsteen, K.; Norman, J. “Designing pervasive games through chunking space-time and mapping between multiple representations” Physical and Digital in Games and Play Seminar. 9th Game Research Lab Spring Seminar, May 29-30, Tampere, Finland


Reenskaug, Trygve “The original MVC reports”, Department of Informatics, University of Oslo, 2007


Pautasso, C.; Wilde, E.; “Why is the Web Loosely Coupled A Multi-Faceted Metric for Service Design”


Stephen, F. “Dashboard Confusion.” Intelligent Enterprise; 2004


Winckler, A.; Palanque, P.; Freitas, C.; “Tasks and Scenario-based Evaluation of Information Visualization Techniques” TAMODIA 2004 Papers, Prague, Czech Republic, 15-16 November


Game references

Ingress, http://www.ingress.com/
The Beast, http://www.cloudmakers.org
Uncle Roy all around you, http://www.blasttheory.co.uk/projects/uncle-roy-all-around-you/
Momentum, http://momentum.sics.se/
Catch Bob!, http://craftwww.epfl.ch/research/catchbob/
Can you see me now?, http://www.blasttheory.co.uk/projects/can-you-see-me-now/
PAC-LAN, http://imagination.lancs.ac.uk/activities/PACLAN
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Information System</td>
</tr>
<tr>
<td>DSR</td>
<td>Design Science in Information Systems Research</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>LARP</td>
<td>Living Action Role Play</td>
</tr>
<tr>
<td>ARG</td>
<td>Alternate Reality Game</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information System</td>
</tr>
<tr>
<td>DBMS</td>
<td>Data Base Management System</td>
</tr>
<tr>
<td>2D</td>
<td>2 dimensional</td>
</tr>
<tr>
<td>3D</td>
<td>3 dimensional</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency Identification</td>
</tr>
<tr>
<td>STC</td>
<td>Space-Time-Cube</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architectures</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>W3C</td>
<td>Wold Wide Web Consortium</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>API</td>
<td>Application Protocol Interface</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>NPC</td>
<td>Non-Player Character</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>SOFEA</td>
<td>Service Oriented Front-End Architecture</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>MUD</td>
<td>Multi User Dungeon</td>
</tr>
<tr>
<td>MOO</td>
<td>MUD Object Oriented</td>
</tr>
<tr>
<td>RSS</td>
<td>Really Simple Syndication</td>
</tr>
<tr>
<td>WAMP</td>
<td>WebSocket Application Messaging Protocol</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
</tbody>
</table>
Source of external figures

Figure 1 http://www.itc.nl/personal/kraak/move/ Copyright ©
Figure 3 Copyright © Mitsubishi Electric Research Laboratories, Inc., 2010
Figure 3 Copyright © The Eurographics Association 2012
Figure 5 http://www.girardin.org/fabien/catchbob/pervasive/ Copyright © Fabien Girardin
Figure 6 Copyright © 2008 Paul Coulton et al.
Figure 8 http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ Copyright © W3C (MIT, ERCIM, Keio)