NTNU – Trondheim
Norwegian University of
Science and Technology

DSL and Engine for Pervasive Treasure Hunt Games

Habibollah Hosseinpoor
Christian Skar

Master of Science in Computer Science
Submission date: May 2012
Supervisor: Hallvard Trætteberg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

The purpose of this master thesis was to eliminate the technical barrier for creating pervasive games. In order to achieve this a domain-specific language (DSL) was made along with an engine to interpret the DSL scripts. This was done by using a customized development method, combining the DSL lifecycle and an iterative game development method. The goal was to make pervasive games more established in society and to make them more popular. Initial stage was to analyze pervasive games and making a game specification. This resulted into the four pervasive dimensions, goal, mobility, social, and temporal, with their three levels, low, medium, and high. These dimensions and levels, along with certain game elements, made it possible to create a DSL expressive enough to make pervasive treasure hunt games. The conclusion was that a DSL can be used to create pervasive games, thus making them more available for the audiences. This could eventually make pervasive games more popular.

Sammendrag

Hensikten med denne oppgaven var å eliminere den tekniske barrieren for å skape pervasive spill. For å oppnå dette ble det laget et domene-spesifikk språk (DSL) med en tilhørende motor for å tolke språket. Dette ble gjort ved å tilpasse en utvikling metode ved å kombinere en DSL livssyklus og en iterativ spill utviklingsmetode. Målet var å gjøre pervasive spill mer etablerte i samfunnet og gjøre dem populære. Innledende forskning var å analysere pervasive spill og lage en spillspesifikasjon. Dette resulterte i de fire pervasive dimensjonene, goal, mobility, social og temporal, med sine tre nivåer, lav, middels og høy. Disse dimensjonene og nivåene, sammen med spesifikke spill elementer, gjorde det mulig å lage et uttrykkskraftig DSL som kan brukes til å lage pervasive rebus løp spill. Konklusjonen var at et domene-spesifikk språk kan brukes til å lage pervasive spill. Dette kan hjelpe til med å øke tilgjengelighet til pervasive spill og dermed gjøre dem mer populær.

Preface

This report is the result of our master thesis written during the spring semester 2012, a total of 18 weeks, at the Norwegian University of Science and Technology in Trondheim. This work presented is based on the two individual projects we completed previous semester, a course named TDT4501 Specialization Project. Our projects had different research focus but complemented each other in the way that they both were meant to support PLAYTRD, a platform for research on pervasive games. We would like to thank our supervisor, Hallvard Trætteberg for his guidance and sharing of expertise during this project.

Trondheim, May 25, 2012

Habibollah Hosseinpoor

Christian Skar

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the introduction to this master thesis. It starts by presenting the motivation and background, before presenting the problem definition. It also contains a readers guide for the reminder of this report.

1.1 Motivation

PLAYTRD is a platform for research about pervasive games. Typical research topics are game concepts, architecture and technology. The purpose with these games is for the players to familiarize themselves with Trondheim by using it as play area. The research we present in this report is meant to contribute to this platform, and we planned to do so by using our skills and competences. We hope our research can contribute in making pervasive games more established in society and making them more popular.

1.2 Background

Imagine that you are walking through the city when you get a message on your smart phone saying; *"WARNING: Zombie outbreak in the city. Get to the metro station at Fifth Street within 20 minutes for extraction. All transportation vehicles are believed to be contaminated, avoid using them at all cost"*. Your body starts to fill with adrenalin as you read the message, and without much delay you start running towards Fifth Street.

After running for several minutes avoiding zombies you feel you need a short break. You run into a narrow alley looking for a place to hide when you see two dumpsters side by side with just enough space to hide between them. Before hiding you look around to make sure that no one followed you or sees you. Safely in your hiding place you look at your smart phone and open an application showing a satellite image of the city. It also shows a timer and an arrow indicating your position as well as the zombies you are trying to avoid.

The metro station is located 300 meters south of your position, and the timer shows there are only 5 minutes left. Knowing that your path to the metro is filled with obstacles and zombies you decide that you've had enough rest. You exit from your hiding place and run towards the end of the alley. Before entering the street you look at your smart phone once again to ensure that there are no zombies in your proximity.

You start to walk casually towards the metro station when all of a sudden your phone starts to vibrate intensely. You know that this is a warning sign that there are zombies very close to you. You can see there are a few people running towards you while looking at their phones. Convinced that you have been spotted you start to sprint towards the metro station. Finally, after an intense chase you reach your destination before any of the zombies were able to catch you, and you receive a message on your phone. You have just won the game and received 100 points.

The scenario above describes an advanced version of a tag game. To be more specific, it describes an example of what could be a pervasive game. The game is played in the city using it as a play area with natural structure and bystanders as obstacles. The goal is to reach a real location in the city and the players either plays the role as a chaser or as being chased. What makes these games differ from other games is that they are tightly interwoven in our everyday lives through the artifacts and people that surrounds us, and the places we inhabit.

If you try to search for pervasive games in a store, such as the one described above, you might discover that there are none. Smart phone applications like *Endomondo* and *Google Latitude* shares some of the same characteristics as pervasive games, but they lack some essential elements. A way to overcome this is by being creative and to establish the game mechanics (rules) separately. The smart phone with the application will then act as an artifact for playing the game, equivalent to what a deck of cards or a dice is for certain games. Regardless of how creative you are, this method adds huge limitations on the game design and we might not be able to create the games we want. In the following sections we try to reveal some of the reasons why pervasive games aren't more popular and bigger than what they are today.

1.2.1 The Community

The first step is to identify who plays and develop pervasive games today. The pervasive game community consist mostly of people with a particular interest and representatives from schools and universities. Searching for these communities on the Internet usually directs you to the same webpage: "Pervasive Games: Theory and Design", who also have provided with a book with the same title[14]. We find this community to be relatively small compared with the conventional gaming communities, e.g. games for PC, handheld devices, and consoles. We are yet to see a greater effort coming from the gaming industry, who possess the necessary resources and technological knowledge to make a difference. They could help to increase the amount, varieties, and quality of pervasive games, which might lead to more and higher demand from the audiences. We can only speculate that their lack of interest is due to the low demand, high risk of failure, and low profit. This

emphasize the importance of the research committed by interest groups, which have to convince both the audience and the game industry that pervasive games can become a commercialized product.

1.2.2 Pervasive Game Challenges

One of the most challenging aspect in pervasive games is the fact that the game uses the real world as play area. When the players are spread over a large geographical area, like the pervasive game example described earlier, makes it difficult to monitor player movement and actions during play. *Ludocity* is a homepage with a collection of pervasive games designed by different authors, and these games uses more or less the same solution for this problem[12]. They have solved this by designing games without the need for constant monitoring of the players. Another possible solution is to design games with a long response time and/or tick based mechanics. This makes time and place independent from each other, making constant monitoring redundant. The drawback with these solutions are that they put huge limitations on the game design and varieties. As a consequence, this might make the games cumbersome and less fun to play, resulting in that few people wants to play them. Another possible solution to overcome the player monitoring problem is to develop software solutions, which brings us to the next section.

1.2.3 Development Challenges

In the latter, new technology has helped to create a platform for supporting pervasive games. Ubiquitous technology like GSM, 3G, smart phones, and Wi-Fi hotspots are some examples. With the use of these technologies it is possible to support pervasive games, and could possibly solve the monitoring problem mentioned earlier. Smart phones have built in technology that can provide real-time information regarding position, data communication, movement, and much more. This makes them the perfect candidates to serve as clients for pervasive games. However, developing software technology introduces some challenges. Figure 1.1 shows a venn-diagram from the article “The future of agile software development” that could help us identify some of these challenges[2].

The venn-diagram shows three overlapping circles representing different success factors; “do right things”, “do things right”, and “timing/Speed”. Together, these circles are combined to make the phrase: “do the right things right and fast” indicating what is required for making any kind of product successful. Furthermore, these overlapping areas can be used to predict the product’s possible faith, depending what circle area the developer is in:

1. **Success** This is where developers should and wants to be.

2. **Miss market opportunities** Does not mean that the product itself is a failure, but the product owner might lose sales due to competitors or the fact that the product is no longer needed or obsolete.

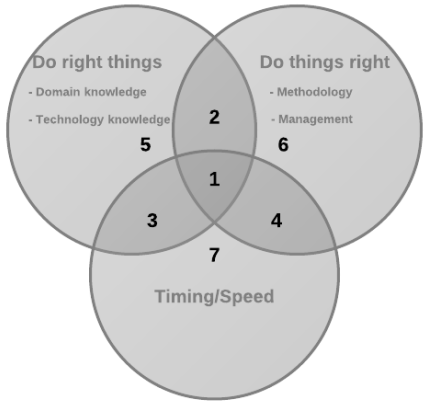


Figure 1.1: "Do the right things right and fast"[2].

3. **Poor quality** The product might be ok, but is likely to have problems in the future.

4. **Failure** High quality product that fails since it is not what the customer wants or need.

5. **Combination of 2 and 3** Could work, but it is highly unlikely that it will be a success.

6. **Combination of 2 and 4** Fails since it is not what the customer wants and/or it is no longer needed.

7. **Combination of 3 and 4** A rushed product with poor quality and not what the customer wants.

As a result from the venn-diagram, Figure 1.1, we ask four questions. By answering them we could help to identify the development challenges.

1. **"What is being developed?"**. This means it is important to identify the needs of the stakeholder, and to establish functional and non-functional requirements for the product. This determines the definition and limitations of the software itself. It is also important that the appropriate technology is chosen in order to fulfill them. This requires good communication with the stakeholders so that the developers know what they need to develop. Furthermore, it requires domain experts so that the right technology is chosen.

2. **"How is the product made?"**. This describes the lifecycle of the product. It is important that development are done in a correct manner in order to achieve a quality product. This requires a project methodology that are suitable for making the product. In addition, this requires trained personnel and managers that know how to use the methodology.

3. **"Who makes the product?"**. It is crucial that the right people are involved in the entire development lifecycle. These are people with expertise within certain domains, such as hardware and software technology, design, marketing, management and many more.

4. **"When should the product be made?"**. This is a difficult question to answer since it is best answered in hindsight. Deploying a product too early might be fatal since the public might not be ready for it yet, e.g. *Dennis Crowley's Dodgeball*, which is the predecessor to *Google Latitude*. However, deploying it too late might make the product obsolete and/or miss market opportunities, e.g. *Duke Nukem Forever*.

By looking at the four questions above, we see that there is a common denominator concerning the people involved in the development. These people need to have the right skills and competences, so that the product has a higher chance of becoming a success. In chapter 1.2.1 we mentioned that the pervasive game community is small. This introduce a particular challenge for pervasive games since it is less likely that the right people are available within their community.

1.3 Problem Definition

We believe that the low demand and the low availability of pervasive games are all factors that are of hindrance for making them more popular. We can see in the gaming industry today, that even thought there is a huge variety and amount of games available, only a small percentage of them that are actually very successful. Fearing that this also applies for pervasive games they might never become popular unless something is done.

We are convinced that this could be solved by creating software. However, when the big game developers are reluctant to do it, someone else needs to take the initiative. With the challenges described earlier in mind, this is not an easy task for someone without any technical background.

There are two questions we would like to study further based on what we have mentioned this far. The questions are divided into two categories, domain, which is pervasive games, and software technology. We would like to investigate what a pervasive game is and to find existing game technologies that are available today. In addition, we want to know if these technologies can be used by someone without any technical background. Based on this investigation we will narrow down our scope and establish our research question.

The following questions guides our preliminary studies:

1. What does it imply that a game is pervasive, and how can we use this answer to make games pervasive?

2. What software solutions are used today for making pervasive games, and is it possible for someone without a technical background to use them for making their own games?

1.4 Report Outline

This section presents a brief overview of the entire document and gives a short description of each chapter.

Chapter 2 - Preliminary Studies It starts with a description of games in general, before gradually moves on to pervasive games. Then it presents game technologies. It also present Domain-Specific Language (DSL) and some associated technologies for DSLs.

Chapter 3 - Project Methodology It starts with presenting the research question, followed by a brief presentation about design-science and how we intend to use it for our research. Then we present a brief description of the DSL lifecycle and iterative game development, which have been combined to make our customized project method. It ends by presenting our toolbox.

Chapter 4 - Results It starts by presenting our findings concerning the domain, before presenting the game specification. Then finally we present the system architecture from a technical point of view.

Chapter 5 - Discussion It starts with a discussion of the project method, then moving on to our treasure hunt DSL. It also contains a discussion concerning the DSL engine and its graphical editor.

Chapter 6 - Conclusion It starts by presenting the conclusion for this master thesis, before presenting our suggestions regarding further work.

Appendix A - Task Contains the initial task description.

Appendix B - Game Concepts Contains eight different treasure hunt game concepts.

Appendix C - First Prototype Documents the first iteration (first prototype), a simple DSL and a simple engine to execute DSL scripts.

Appendix D - Second Prototype Documents the second iteration (second prototype), improved DSL, to support GPS positioning, and the engine reflects to the changes too.

Appendix E - Third Prototype Documents the third iteration (third prototype). Implement support for pervasive dimension in both the DSL and engine.

Appendix F - Fourth Prototype Documents the fourth iteration. Implement support for graphical editor, in order to create game instances, without external IDE or text editor.

Appendix G - User Manual User manual for the last prototype.

Chapter 2

Preliminary Studies

This chapter presents the preliminary studies. It starts with a description of games in general, before gradually moving to pervasive games. Then it presents game technologies. It also present domain-specific language (DSL) and some associated technologies for DSLs.

2.1 Games

"Games lubricate the body and mind"
- Benjamin Franklin

Before we go deeper into what a pervasive game is we need to get more information about games in general. Humans are introduced to games from the very first months of their lives. Parents play games with their children starting with games like *Peek-a-boo*, and later with *Hide and Seek* as the child gets older and more mobile. Playing games seems to be in our nature since they have been an important part of human society since the early stages of civilization. The oldest board game found, *Senet*, has been dated back to 3100 BC and was played by the ancient Egyptians[18].

2.1.1 What is a Game?

A game is structured playing and is associated with enjoyment and fun. In addition, they can be used as an educational tool and to encourage physical activities. Games should not be considered the same as work. However, there are cases where they overlap, e.g. professional athletes/players in sports, poker, chess, Starcraft and many more. These people make a living playing the games normal people play for fun[31].

Games come in many varieties and we have chosen to categorize them into the following forms; sports, tabletop games, video games and other. In order to understand what these categories implies we are going to describe them further and mention some game examples for each form:

Sports are physical activities that aims to entertain the participants. This is done by using, maintaining, or improving their own physical fitness and strategical thinking. A sport can be played casually, or through organized settings, e.g. olympic games, world games, X games, where you can compete against other participants. Example sports are; football, javelin, sprint, tennis.

Tabletop games are games usually played on a table or any flat surface. Typical games are board games, card games, and dice games. This form represents the typical form of games that have been played since ancient times and still played today. However, it seems that this form of games is decreasing due to succeeding form. Some example of such games are *Ludo*, *Poker*, *Monopoly*, and *Trivial Pursuit*.

Video games are a multi-billion dollar industry today, and represents games that are played on platforms such as handheld devices, PC's, and consoles. The players interact with the game through a user interface that generates a visual feedback to the player. Typical games are *Super Mario*, *Counter-Strike*, *World of Warcraft*, and *Skyrim*.

Other games are games that does not necessarily fall under the forms mentioned above. Some examples are street games, drinking games, and war games.

Furthermore, the forms can be broken down into game genres, which can exists across the different forms of games and platforms. Table 2.1 shows the game genres that are most common for describing games today.

	Sports	**Tabletop**	**Video games**	**Other**
Action	Paintball	Warhammer	Call of Duty	-
Adventure	Geocaching	-	Monkey Island	Letterboxing
Role-playing	Hauk og due	D&D	Skyrim	Live RPG
Strategy	Chess	Risk	Starcraft	-
Puzzle	-	Sudoku	Tetris	Rubik's cube
Simulator	Skeet	Income outcome	Americas Army	-
Children's game	Tag	Tic-Tac-Toe	Super Mario	I spy

Table 2.1: Game genres in different forms

2.1.2 Why do we Play Games?

We have already mentioned that people play games for the sake of enjoyment and fun. XEODesign, a consultant firm for putting emotions into play, has committed a research concerning "why we play games". This has resulted into four keys of emotions[11].

1. **Hard fun** is for people that enjoy challenging tasks, strategic thinking and/or problem solving. This could be associated with the feeling of accomplishing something difficult. Typical games that fall under this emotion are single-player games like *Tetris*, *Sudoku* and *Solitaire*.

2. **Easy fun** fulfills intrigue and curiosity. This can be associated with a game plot. It can sometimes be hard to put a side a video game that has an appealing plot that is revealed as you play. This awakens the player's curiosity, thus making them want to continue to play. Video games like *Uncharted*, *Metal Gear Solid*, and *Fallout* are know for having good plots and stories.

3. **Altered states** makes people play for the internal sensation such as excitement and relief from their own thoughts and feelings. Sometimes it might be appealing to "disconnect" from the reality. This could be due to a stressing environment or the need to think about other things. In most cases this applies for all games.

4. **People factor** addresses the social aspects around and within a game. Competition is a huge driving force for many people. New video game genres like MMORPG's have created a new medium for social interaction between people from all over the world. Typical games that have this emotion are *World of Warcraft* and *Counter-Strike*.

2.1.3 What Defines a Game?

There are three aspects that are necessary in a game; mechanics, dynamics and play. The game mechanic is defined as the game rules. The dynamics describes the pattern, or behavior of a game, like saying that the video game *Ping Pong* has the same dynamics as tennis. The game play describes the interaction between the players and the game rules. In order to explain the game play better we have used the definitions defined by Roger Caillois in the book *Man, play and games* [1]. According to Caillois, there are four game play categories, see Table 2.2. In addition, there are two play types, *paida* and *ludus*, which are structured to unstructured play respectively, see Table 2.3.

2.1.4 Pervasive Games

The word pervasive is defined as to *"having the quality or tendency to pervade"*. A television with it's channels and a device with Internet are good examples of pervasive objects that are used by many today. As a consequence, they influence our culture and can be perceived outside their own domain.

> *"Pervasive games are new game experiences that are tightly interwoven with our everyday lives through the items, devices and people that surround us and the places that we inhabit"*[14].

Play categories		
Name	**Description**	**Game examples**
Agon	To compete, either against yourself, against opponents direct or indirect.	Chess
Alea	Random, there are situations in the game that the player has no control over.	Poker, roulette
Ilinx	Also called vertigo, is the sense of disrupting your own perception.	Tag game, hide and seek, sports
Mimicry	To imitate someone or something.	RPG's like Dungeons and Dragons, simulators

Table 2.2: Game play categories

Play types		
Name	**Description**	**Game examples**
Paida	Structured, the game follows a strict structure giving little or no freedom to the players	Racing games
Ludus	Unstructured, the players are allowed to do almost everything and can decide the order of events themselves.	Grand Theft Auto

Table 2.3: Game play types

According to *TeMPS*, which is a conceptual framework for pervasive and social games, there exist four perspectives concerning pervasive games[6]. These are represented in the name *TeMPS*; temporal, mobile, perceptual and social. Temporal is related to the time a game is played and the duration of it. In addition it says something about how important time is for the game, e.g. if time is critical or not important at all. Mobile is associated with the play area, which defines where the game is played. It can also tell us something about how the game is played by identifying how the players interact with the game world. Note that the game world is the real world mentioned earlier. The perceptual aspects addresses how the real world is perceived in the game. The social perspective means the direct and indirect interactions between players and non-playing persons.

2.1.5 Game Prototyping

A prototype is a temporary version of a product. A typical prototype don't have all the features and functionality that the final product possess. Their details and complexity depends on the resources available, and are usually divided into two categories; low-fidelity (simple and cheap) and high-fidelity (closer to the final

product). In addition, the prototype is either horizontal or vertical, which describes the implementation of features and functionalities respectively[19].

Prototypes are good for evaluating games early in the development. They can provide with valuable feedback, either through the development itself or through testing. This will help to shape the game in accordance with customer demands and needs. In the article "Using Prototypes in Early Pervasive Game Development" the authors addressed three types of prototyping methods; rapid game development, prototyping with ready made software, and physical prototyping[15]. Furthermore, they conclude that prototypes should be made as early as possible with the correct method. The choice of method should depend on certain criteria, which can be seen in Table 2.4.

Game type	**Prototyping method**
Context-aware (sensor input needed)	Often easier to implement as a software prototype; Wizard of Oz prototyping is a good alternative.
Discrete (events occur in predictable manner)	Physical prototypes as well as software prototypes.
Continuous (e.g., events are functions of location and other sensor input)	Software prototype is useful.
Technically innovative	Software and/or hardware should be used early to test technical aspects.
Social novelty	Real users should be involved in realistic situations. Both software and physical prototypes can be used, e.g, Wizard of Oz prototyping or para typing. Can also be supported with interviews, focus group discussions, and ethnographic studies.
Complex interaction between various gaming platforms	Can be difficult to demonstrate with physical prototypes.
Persistent, long-term	Software prototypes or prototypes with software components are good. Testing with physical prototypes is difficult but can be useful in testing core mechanics.
Player-to-game interaction: dexterity based games	If manipulating game objects physically is central in the game, as in dexterity based games like *Tetris*, software prototype is needed.

Table 2.4: Selecting prototyping methods based on game type

2.2 Game Technologies

This section starts by introducing the technological aspects for games. It continues by introducing customized software packages that are used for making context-aware games, before introducing a few pervasive game examples that have used software to support their games. It continues by exploring relevant research and technology. All these sections tries to explain the technology used, and how easy they are to use for people without non-technical background.

2.2.1 Video Games

In the late 1940s the world was introduced with the first video game through the cathode ray tube-based missile system, setting the starting point for the history of video games. Since then video games have gradually increased in both sophistication and complexity, and has eventually diverged onto different platforms such as PCs, consoles, and handheld devices.

A video game is the result of a symbiosis between hardware and software technology. The game mechanics and dynamics, as mentioned in section 2.1.3, are implemented as software code. The player interacts with the game through hardware devices such as a joystick, keyboard, or a hand controller. A visual feedback is returned to the player either through a television, PC monitor, or the screens on the handheld devices, depending on what platform the game is played with. In addition to this, ambient music and sound effects are implemented to make the games come more to life. All these features requires a multidisciplinary work force, like having programmers, sound engineers, graphical designers, and more, working together to create a game[30].

2.2.2 Game Engine and Level Editor

A car without an engine doesn't work as a transportation device since it wouldn't have any propulsion system. The same applies for games if there were no engine to handle the player input in the game. Furthermore, a car engine can be taken out and used as a propulsion system for other cars, depending if it fits or not. This can also be done for game engines. A game engine is a software framework that abstracts the details of doing common game related tasks, such as rendering, collision detections (physics), sound, artificial intelligence, and networking[29]. The main purpose with game engines are that they can be reused to make many different games. Choosing to use an existing game engine the developers can save both time and money.

Some engines provide with an integrated development environment (IDE) for programmers to create games. The most commonly used game engines today are *Unreal Engine*, that got their own scripting language (UnrealScript) and editor (UnrealEd), *CryEngine* and *RAGE Engine*[22]. However, these engines are licensed, which imply that the developer needs to pay the engine owner if the engine is used. In addition to these game engines there exists hundreds of others, ranging from a simple programming library to high-level, both open-sourced and licensed[33].

Game engines does not completely eliminate the technical barrier for creating games, as Figure 2.1 illustrates. Using an existing game engine puts limitations on the game design, since the game can't exceed the engine's capabilities. At the same time, it reduces some of the complexity for developing a video game.

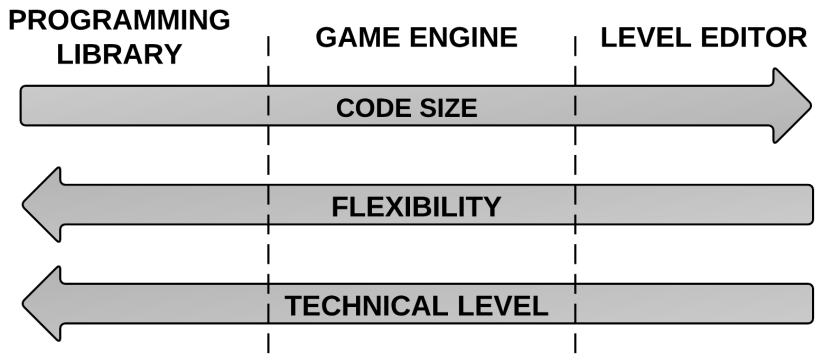


Figure 2.1: Code size, flexibility and technical levels for game engines and level editors

A level editor is a software tool used for designing levels, maps, campaigns, or similar, for a specific game or game engine[32]. These tools require a lot of time to be created due to the large code size. Their expressiveness is very limited, meaning they are very restrictive with what they are capable of. However, they don't require any programming knowledge to be used. As a consequence, a level editor doesn't require the user to have a technical background.

2.2.3 IPerG Software

IPerG is short for integrated project on pervasive gaming. It was a EU funded project that started in 2004 and ended in 2008. The IPerG project have created three software packages for creating and staging pervasive games. These three packages offer different solutions that can be used depending on a game's technical standpoint[24].

1. **Augmented-/virtual Reality** package includes the Morgan AR/VR platform (MAP), Logfile Analysis tool (LAT) and the Polling tool.

2. **Mobile Phone** based game solution package for games that uses the mobile phone as an input/output device, supported by a server. The main software component is the MUPE platform but also introduce the Web Application Framework (WAF).

3. **Ubicomp** based game solution package that targets games ubiquitous computing principles, embedded processors and the use of sensor and actuator

hardware. It uses a Java gaming library, based on the PIMP and PART platforms, and in addition the Logfile Analysis tool and the Polling tool.

In order to understand more about these packages we need to identify each software component, see Table 2.5.

SW Component	Short explanation
MAP	Morgan is a commercialized licensed product that supports the development and the usage of augmented reality and virtual reality applications.
LAT	is a tool for analyzing events that occur during the lifetime of an application.
Polling tool	is for making online (web) questioners to capture player experiences for post-game evaluation.
MUPE platform	is short for Multi-User Publishing Environment. It is a platform for mobile multi-user context-aware applications and uses a plethora of technologies for reducing the complexity in developing applications for mobile services. MUPE is available under a *Nokia Open Source License* (NOKOS).
WAF	supports the development and integration of java-based web applications. It is available as open source under the BSD license.
PIMP platform	is a java library that can be used to build distributed applications.
PART platform	is a light-weight distribution and data sharing library with focus on handheld and embedded devices and peer-to-peer connectivity. It is available under the BSD license.

Table 2.5: Software components in the IPerG software packages

The IPerG software packages offer good solutions for making pervasive games. However, they are meant for software developers, which makes them less appropriate for people without the proper knowledge to use them. This doesn't mean they can't use them, but it would require extra time and effort just to learn the technology and how to use it.

2.2.4 Pervasive Game Examples

GeoQuiz

GeoQuiz is an example of a pervasive game developed with an IPerG package. It is a mobile location-based question and answering game. The players themselves make questions that are linked to physical locations and answer the questions that have been made by other players. The game is played indefinitely, which implied that no player will actually win the game. However, there are high score lists based

on the number of correct questions answered and locations found that provides a competitive element in the game.

The technology used in this games are built in the mobile clients and uses Java (J2ME). In addition, they use *Placelab*, which is a Symbian application that allows java applications to see what GSM mast the phone is connected to[24].

Pac-Manhattan

Pac-Manhattan is a recreation of the 1980s video game called *Pac-man* with a few essential changes. The game uses the New York City grid as play area. The players runs around dressed according to their role either as Pac-man or as a ghost. The game mechanics are the same as in the video game, where Pac-man have to collect "virtual" dots while avoid getting caught by the ghosts. This game is supported by using cell-phones, Wi-Fi Internet connection and a custom software developed by the Pac-Manhattan team[25].

The specific software used by the Pac-Manhattan team is unknown, but we know there is another *Pac-man* game that bases its software on theirs. The game is called *The Urban PacMan* and is made by the PacMan@Lyon team. No installation on the client side is required, meaning the cell-phones, but the server requires a web server with a PHP interpreter and a database. They recommend a classic configuration using XAMPP (Apache, MySQL and PHP)[17].

2.2.5 Authoring Tool

The article "Designing Enhanced Authoring Tools for Pervasive Games"[28], describes a proposal for a framework. The purpose is for allowing non-technical game designers to create pervasive games using a GUI-based tool. Their pervasive gameplay model is based on the *TeMPS* framework, which they have used for extracting atomic game elements and are combined in order to make games. Further it explain how the user interacts with the authoring tool for making the games.

The application is based on the *OpenBlocks* library, which enables the developer to build and iterate their own graphical block programming systems by specifying a XML file. However, a compiler/interpreter is required for the programming language created with the block editor. This will require a lot of time and effort, but is necessary in order to create a tool with greater flexibility.

This framework is a good start towards making a tool for creating pervasive games. However, there are still much work needed before this authoring tool can be realized. The major challenge with this idea is to make the implementation robust enough so that the designer doesn't have to worry about the complexity of the underlying technologies. In addition, it raises some questions about how expressive and what limitations that are present with the defined gameplay model.

2.2.6 Complexity in the Software Industry

There have always been a need for reducing the complexity in computer science. In the very beginning of software history the programmers developed software ap-

plications using a low-level programming language like machine code and assembly language. This required highly trained and expert programmers due to the low abstraction level in these languages, and had long expensive development cycles. This was a huge drawback since this might have made it cheaper to hire people to do manual labour instead of paying for an expensive computer program that does the same work. In order to overcome this it was necessary to reduce the syntactical complexity by raising the level of abstraction, which would make software development easier and cheaper. As a result, high-level programming languages emerged[20].

Today, very few software applications are built from scratch, but rather built on other frameworks. However, given the number of different frameworks, libraries, and the communication between other applications we face a new kind of complexity. In contrast to the beginning we now have an abstraction level that is very high. This makes it possible to develop more or less anything we want, but at the same time raises the level of software complexity. It is likely that you wont need all the frameworks that a programming language offers. If you only need to know a few frameworks, why should you then need to know all the others? This question has been asked and answered before, and have led to languages such as HTML, SQL, and BNF. These are so-called domain specific languages (DSL) that are dedicated to a particular problem domain. This represents the solution for making programming languages less complex for the user, and will be further explained in chapter 2.3.

2.3 Domain Specific Language

This section present the concept of domain-specific language (DSL). It starts by providing some definitions, followed by pointing out the benefits and disadvantages of DSLs, and finally a small discussion about different types of DSLs.

2.3.1 What is a DSL?

Figure 2.2, shows the two approaches in all branches of science and engineering that we can distinguish, namely generic and specific [27]. Generic approaches attempt to provide a solution for a broader problem domain, but such a solution may not be optimal. On the other hand, a specific approach provides a better solution, but for a smaller problem domain. These approaches can be directly linked to the topic of generic programming language (GPL) vs domain-specific language. A GPL that are turing complete, such as Java, can be used to solve any kind of programming problems, but may not always be the best tool to use. A DSL on the other hand is designed for a specific problem domain and can solve them better.

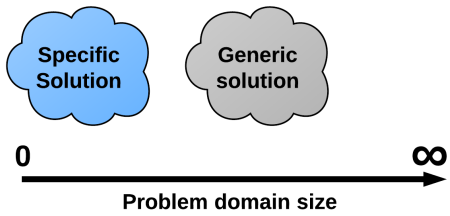


Figure 2.2: Generic solution vs Specific solution

Before we discuss any further about DSLs we need some definitions in order to get a better picture of what it is. However, there are no consensus definition of what a DSL is, we therefore present three different definitions:

1. *"A computer programming language of limited expressiveness focused on a particular domain"* [10].
2. *"A domain-specific language is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain"* [27].
3. *"A domain-specific language is a language designed to provide a notation tailored toward an application domain, and is based only on the relevant concepts and features of that domain"* [4].

Common for all of these definitions are the following; (1) a DSL is a programming language, (2) it provides expressive semantic power in their domains, and (3) it is restricted to a particular problem domain.

2.3.2 Domain Size vs Semantic Expressiveness

Given a language L, we can express the semantic expressiveness, or level of abstraction, of language L as *expressiveness(L)*, and the problem domain size that the language are able to solve as *domainSize(L)*. It seems that the following formula applies in general, but we can't prove its correctness:

$$domainSize(L) * expressiveness(L) = Constant$$

This gives us a nice curve, see Figure 2.3, where we can place both DSLs and GPLs on it. Java has a higher value of domain size but on the other hand low value of semantic expressiveness. The exact opposite applies for DSLs such as SQL. However, the formula does not always apply. Take for instance a language that are specific to a domain, but at the same time can have low value in semantic expressiveness. This is due to the fact that a DSL is only based on the relevant concepts and features of a particular domain.

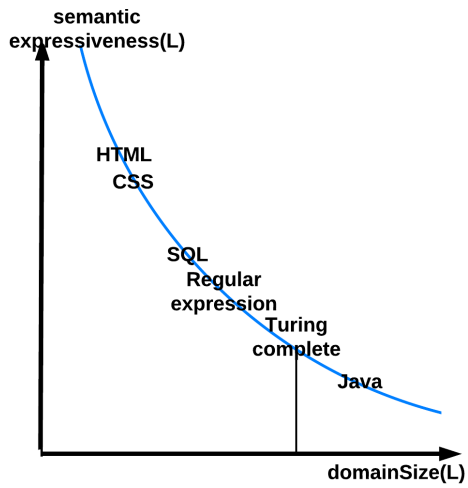


Figure 2.3: Domain size vs semantic expressiveness

2.3.3 Advantages and Disadvantages with DSLs

We have already mentioned some of the advantages with DSLs, which was that they can solve their problem domain better than GPLs. In addition, there are other advantages and disadvantages, which needs to be elaborated further.

Increased development productivity, DSLs allows solutions to be expressed in a high level of abstraction within the problem domain, which allow us to write clear and concise code, that is self-documenting to a large extend, due to the DSL's domain specifics. *Communication with domain experts*, since DSLs are designed for a particular problem domain they use common language, terminologies within the problem domain, which means the same thing to everyone with respect to

the problem domain. This allows for much more efficient communication with the domain experts, which in turn allows the domain experts themselves to understand, validate, modify and often even develop DSL programs. *Shift in execution context*, DSLs are usually declarative and can therefore be viewed as specification languages, as well as programming languages. Due to DSLs decorativeness it allow us to shift its execution context very easy. For instance for a DSL program, we do not care about how it is executed as long as it is executed according to the DSL's specification. This allows us to switch between different execution engines and platforms.

The main disadvantage with DSLs is the cost of building. It requires both domain and language development expertise. For instance, developing a DSL that is only readable by a domain expert is much cheaper than a DSL that are writable by other domain experts. In addition, they are also hard to design due to the challenges in finding the proper scope for a DSL.

2.3.4 Different Kinds of DSLs

DSLs can be categorized by different means/dimensions, such as (1) level of executability, (2) whether it depends on an existing language, (3) the notation used, or even (4) the level of expressiveness (domain coverage). The last category is not very common, since the domain that a particular DSL covers can have a different domain size.

Category 1, namely the level of executability, is simply drawing the line between DSLs that are executable and DSLs that are not executable. Category 2, draws the line whether the DSL is based or has dependencies on an existing language, which is called a base language. External DSLs, are usually built from the ground up, allowing the developer to create custom syntax and concepts that are closely related to the domain. On the other hand the costs for development compared to an internal DSL are huge. Internal or embedded DSLs uses a host language, so that the internal DSLs syntax is both influenced and restricted by the host language selected. Depending on the selected host language, it could be simpler to build one. Think of this kind of DLSs as a good application programming interface (API) development. Category 3, determines if the DSL has a textual or a graphical notation. The most common DSLs in existence today have textual notations, such as SQL and HTML.

Figure 2.4 shows examples of different kinds of DSLs. DSL **A** can solve the same problem as DSL **B** due to the fact that **A** has a broader domain size than **B**. However, this doesn't apply the opposite way since **B** has a smaller domain size than **A**. There can also be DSLs that overlap each other, such as **A** and **C**.

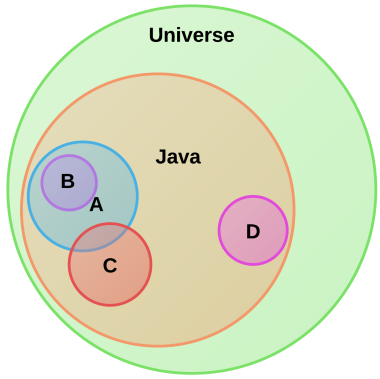


Figure 2.4: DSL domain and level of expressiveness

2.4 DSL Technologies

This section presents tools and technologies involved when creating a DSL.

2.4.1 Tools for Creating DSL

As mentioned in section 2.3.4, DSLs can be categorized based on its dependencies to existing (host) languages. Creating internal DSLs are simpler because you do not need to build parsers/compilers external to your code. Internal DSLs may not be as expressive and readable to non-developers as external ones that can use natural languages such as English. Furthermore, creating an internal DSL can be compared with creating APIs. This means you do not need special tools to create them, but when it comes to readability it will dependent on the host language. Some languages are more suited as host language for internal DSL than others, such as Lisp.

External DSLs have their own custom syntax, therefore you need a full parser/-compiler to process them. It can be implemented either by interpretation or code generation. Interpretation, is simply reading in the DSL script and executing it at run time. Code-generation, on the other hand is to read the DSL script as input and produce code that is itself a high level language, such as java. Figure 2.5 illustrates the difference between these two implementation strategies.

Tools for Creating External DSLs

When creating external DSLs one can either build everything (parser/compiler) from scratch, or use tools that are designed to reduce this workload. We will present two popular tools for creating DSLs, one that have a graphical syntax, *DSL Tools*, the other a textual syntax, *Xtext*.

Microsoft's Visual Studio provides with *DSL Tools* and a text template framework [26] for creating DSLs. The *DSL Tools* is used for building custom designers for visual domain-specific languages, and the text template framework can be used to write code generators. Together, they can be used to produce custom tools to support domain-specific software development, where DSLs are used to create models that provides the correct input to drive code generators using text templates.

Xtext is a framework for developing programming languages and domain specific languages [34]. It covers all aspects of a complete language infrastructure. It is a powerful tool for creating external DSLs using a textual notation. Figure 2.6, shows the three main steps required when working with *Xtext*. The first two steps are used for defining a DSL and the tools for writing models/instances with this DSL. The third step is for executing the DSL scripts. The last (fourth) step is only required if the engine requires a graphical user interface (GUI).

1. **Define the grammar**. *Xtext* provides with the tools for grammar definition.

2. **Run the generator**. Once the grammar is in place, *Xtext* will derive various language components by running the provided generator that will trigger the

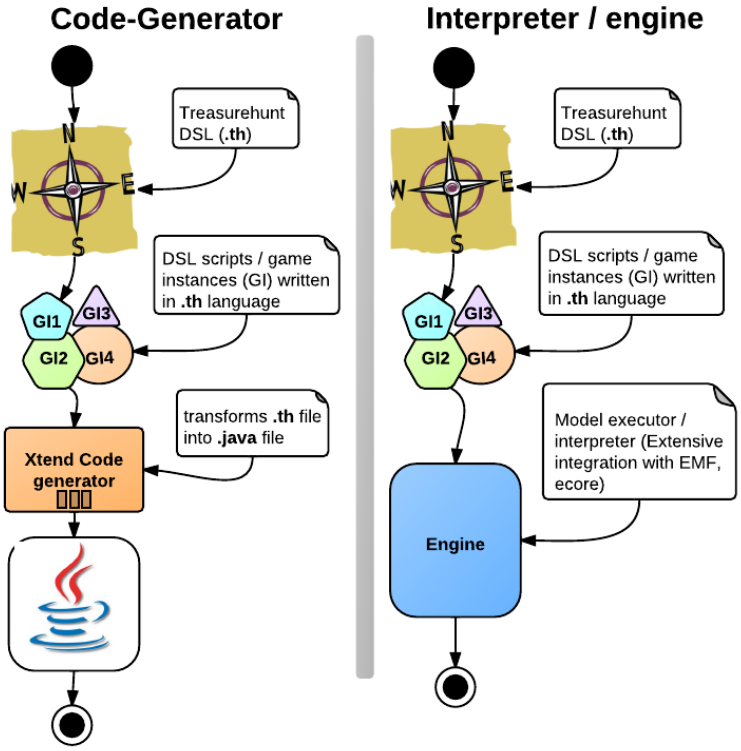


Figure 2.5: Code-generator vs Interpreter

Xtext language generator. It generates the parser and serializer and some additional infrastructure code, such as an editor for writing models/instances in the newly created DSL. The editor is very sophisticated, it comes equipped with code completion, syntax coloring, custom keyword coloring, real time constraints, validation and more.

3. **Define/write the engine**. Once the language is created, we need to know how to execute the models. In *Xtext*, models are implemented using the *Eclipse Modeling Framework* (EMF), which can be seen as a powerful version of *JavaBeans*. There are two main approaches, using a code generator or using an engine or interpreter.

4. **Edit the GUI**. (optional) If a GUI for the engine is required, one need to update it as well in order to reflect to the latest functionalities.

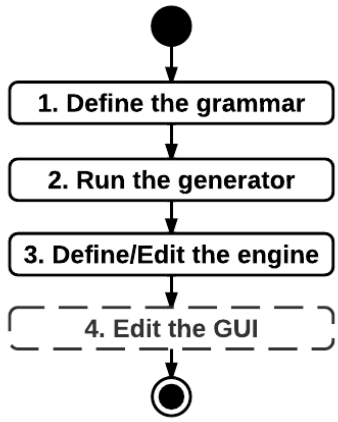


Figure 2.6: Xtext - DSL development steps

Chapter 3

Project Methodology

This chapter starts by presenting the research question, followed by a brief presentation about design-science and how we intend to use it for our research. Then we present a brief description of the DSL lifecycle and iterative game development, which have been combined to make our customized project method. It ends by presenting our toolbox.

3.1 Research Question

Through our preliminary studies we managed to answer our initial question, that asked what it implied that a game is pervasive. We started by exploring games in general moving towards pervasive games, which helped us to define different forms of games, what defines them, and prototyping techniques. We find the *TeMPS* framework to be a good tool for evaluating and creating pervasive games.

The second question concerned about the game technologies that are used to develop pervasive games today. There exist many programming languages and technological devices that could be used to support pervasive games. We found that IPerG were the ones that have had the most progress. However, their software packages were intended for software developers, which makes them less appropriate for someone without a technical background. We felt that the paper presenting the authoring tool had a great idea. In addition, they used the *TeMPS* framework for creating the gameplay model. We also think that a DSL and engine could be a possible solution for creating pervasive games. This have led to the following research questions, which will guide the rest of this master thesis:

RQ1 Can pervasive games be broken down into smaller elements in order to combine them to make new and existing pervasive games?

RQ2 Can we make a DSL with the proper level of expressiveness in order to make pervasive treasure hunt games?

RQ3 Can the DSL be used outside its purpose to make games that the DSL was never intended for?

3.1.1 Short Explanation

The research for this thesis focus on two areas; domain and software technology. The domain represents pervasive games. The second area is software technology and represents how we want to reach the goal mentioned in section 1.1. Therefore, we find it natural that the research question also is divided into domain, **RQ1**, and software technology, **RQ2**. In addition, we want to answer a question that involves both areas, which is **RQ3**.

3.1.2 Limitations

Due to project limitations it is very unlikely that we are able to cover all pervasive games. As a consequence, we will provide qualitative research by having only one focus area that concerns treasure hunt games. However, we hope that the selected game consists of elements that are repeated in other games, thus identify the same aspects for other games. First, we need to know more about what a treasure hunt game is, what defines them, and to find examples of pervasive treasure hunt games that exist today.

Treasure Hunt Games

A treasure hunt game is a game where the player tries to find the secret location of a treasure. This is usually done by following a set of clues where the last clue reveals the treasure. The game can be played by one or many players, and can be played both indoors and outdoors.

Most people are likely to associate treasure hunt games with their childhood or simply as a children's game. A typical scenario where treasure hunt games are played are during children's parties. Before the party starts, a parent hides a series of notes in a certain area. These notes will contain typical clues like "What gets wetter as it dries?", which indicate that there is a note with a clue hidden in a towel somewhere. The final note will reveal the location of the treasure, which in this case is likely to be candy for the kids attending the party.

Treasure hunt games are played by adults too. They can be used as team building and/or as ice-breaker exercises for companies, schools, or for other events. There exist advanced forms for treasure hunt games, like *Geocaching* and *Letterboxing*. *Geocaching* is described as an high-tech form of hide and seek, but shares many characteristics with a treasure hunt game. By using a global positioning system (GPS) the player can find "geocaches" that are spread throughout the world. These geocaches contains a block where the player marks his unique player-ID similar to orienteering. *Letterboxing* is similar to *Geocaching* except the use of art and puzzle's, in addition to leaving your player mark.

3.2 Design Science

> *"Design science seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished"* [7].

3.2.1 What is Design Science?

Design-science is a paradigm based on engineering and science of the artificial. It can be used as a process for recognizing, defining, and solving problems. Design-science is also an interdisciplinary study since it introduces foundations and methodologies used by many disciplines. A reason for this is the fact that an artifact is bound by natural laws and behavioral theories. This means that future artifacts will share the same boundaries as the existing ones. In design-science you focus on the goal and solution simultaneously.

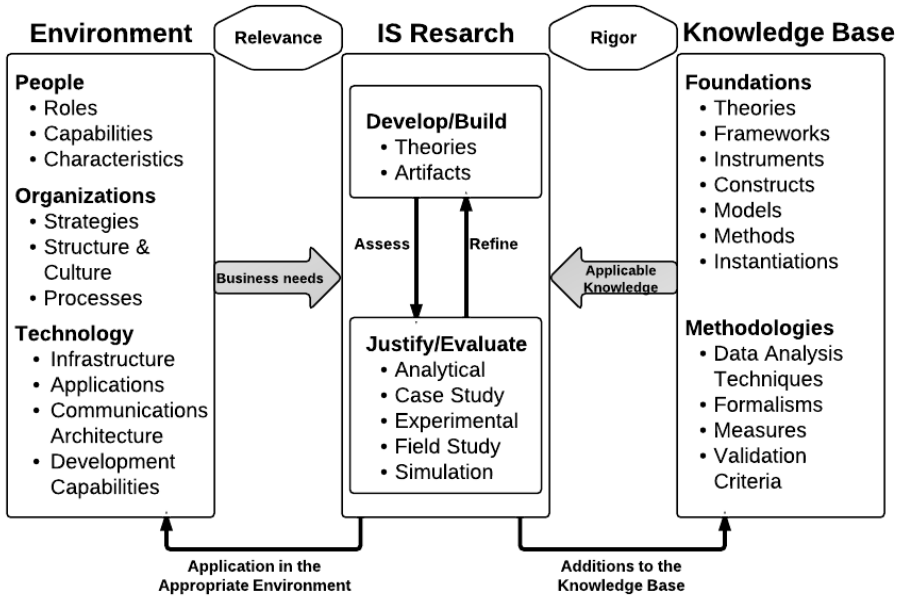


Figure 3.1: Information Systems Research Framework [7]

3.2.2 Design Science and our Project

The concept behind pervasive games might have been introduced many years ago, but there are still many aspects of it that remains unexplored. It is therefore suitable to use a design-science approach for this master thesis. This will be done by

Guideline	Description
Guideline 1: Design as an artifact	Design science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluations of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 3.1: Design-science research guidelines [7]

following a set of guidelines that has been created especially for research concerning information systems, see Table 3.1. The goal for this project is to provide with research about software technology that will support pervasive games. The path we have chosen was an iterative process, which is further explained in the succeeding chapters.

Guideline 1: Design as an Artifact There are three artifacts created during this project; a domain specific language, an editor for creating models, and an engine for interpreting the models. The DSL and engine will be our main focus areas since these artifacts are gonna help us to reach our goal. The editor will be made purely for the sake of supporting and completeness.

Guideline 2: Problem Relevance In order to create games that uses ubiquitous technology, research concerning software technology is required. This introduces new challenges since little has been done by the big gaming industry, which emphasize the importance of research provided by interest groups with few/low resources. The problem lays therefore in the technological complexity, which designers face when developing software for pervasive games.

Guideline 3: Design Evaluation The DSL will be evaluated on its expressiveness, which implies how many different types of treasure hunt games it is capable to create. This will be done by using mathematical methods. In addition, it will be evaluated based on its ease of use and if the DSL can be used outside its purpose. The engine will be evaluated based on whether it is able to interpret models created with the DSL. This will be done by testing and analyzing.

Guideline 4: Research Contributions The design artifacts - the DSL for creating pervasive treasure hunt games developed during this master thesis is our main research contribution. The DSL is excerpted from *Xtext*, thus little is being contributed in terms of a new syntactical language. However, there are some contribution in terms of how to use and structure the code for making the pervasive games. In addition, we will determine if the method developed during this thesis, see section 3.5 is a sound method for creating and evaluating such an artifact.

Guideline 5: Research Rigor Methods for construction of the artifact have been applied in all parts of the development. The methods are discussed in the following sections 3.3, 3.4 and 3.5 in this chapter. Furthermore, we have used the *TeMPS* framework for establishing game patterns. Methods for evaluation - mathematical methods are used for evaluating the expressiveness of the DSL.

Guideline 6: Design as a Search Process This is reflected in the method we have used, which follows an iterative process where the DSL and engine is evaluated at each iteration. They are evaluated against the requirements and goals we have established, and further actions are based on this evaluation. More information regarding the requirements can be found in section 4.3.

Guideline 7: Communication of Research Technology-oriented - which need sufficient details about the artifact constructed. This has been solved by adding detailed information like code, technology and other relevant information in the appendices. Management-oriented - which in this context are interest groups for pervasive games. This means we will have to try to "sell" our research, making them eager to use or continue our research. In addition to this, we wish to speak to a larger audience by catching their interest and to enlighten them about pervasive games.

3.3 DSL Lifecycle

We have decided to use the development method provided by [13]. This is a DSL life cycle, consisting of five development phases; decision, analysis, design, implementation, and deployment. However in practice DSL development is not a sequential process.

3.3.1 Decision

This is the starting phase of DSL development, where the developer has to decide one of the following options in order to solve a particular domain problem:

1. Develop a new DSL
2. Reuse existing one
3. Use GPL

Option one, developing a new DSL, is not recommending when the domain in question is fresh and little knowledge is available, thus option 2 and 3 is more suited.

3.3.2 Analysis

This phase consists of identifying the problem domain and gathering knowledge. More specifically the developer need knowledge from sources such as; technical documents, domain experts, existing GPL code (code base) and customer surveys, in order to produce the following:

1. A domain definition, defining the scope of the domain.
2. Domain-specific terminology (vocabulary)
3. Domain-specific semantics in abstract form.

3.3.3 Design

[13] have identified that DSL design can be characterized along two orthogonal dimensions;

1. The relationship between the DSL and existing languages.
2. The formal nature of the design description.

The first dimension is about choosing either language invention or language exploitation for designing a DSL. If the DSL is based on language invention, the DSL is created from scratch with no commonality with existing languages. If the DSL is based on language exploitation, by using an existing language, there are three options left for designer to decide:

1. **Piggyback**, existing language is partially used.
2. **Specialization**, existing language is restricted.
3. **Extension**, existing language is extended.

Once the relationship to the existing languages have been determined, the designer has to specify the design before implementation. They distinguish between informal and formal designs. Informal design means that the DSL are described informally, e.g. a DSL using a natural language. Formal design means that the DSL are described formally by using an existing semantic definition method, such as regular expressions or grammars for syntax specifications.

3.3.4 Implementation

For executable DSLs they identify the following implementation patters, which a DSL developer has to choose from during implementation:

1. **Interpreter** DSL constructs are recognized and interpreted using a standard fetch-decode-execute cycle. With this pattern no transformation takes place, the model is directly executable.
2. **Compiler/application generator** DSL constructs are translated to base language constructs and library calls. This is also called a code generation.
3. **Preprocessor** DSL constructs are translated to construct in an existing language (the base language). Static analysis is limited to that done by the base language processor.
4. **Embedding** DSL constructs are embedded in an existing GPL (the host language) by defining new abstract data types and operators. A basic example are application libraries. This type of DSL is mostly called an internal DSL, as mentioned before.
5. **Extensible compiler/interpreter** A GPL compiler/interpreter is extended with domain-specific optimization rules and/or domain-specific code generation.
6. **Commercial Off-The-Shelf** Existing tools and/or notations are applied to a specific domain.
7. **Hybrid** A combination of the approaches mentioned above.

Furthermore, they have provided with a decision diagram on how to proceed with DSL implementation, see Figure 3.2, that shows when a particular implementation approach is more appropriate.

3.3.5 Deployment

In the deployment phase the DSLs and the applications constructed with them are used. Developers and/or domain experts use the DSLs to specify models.

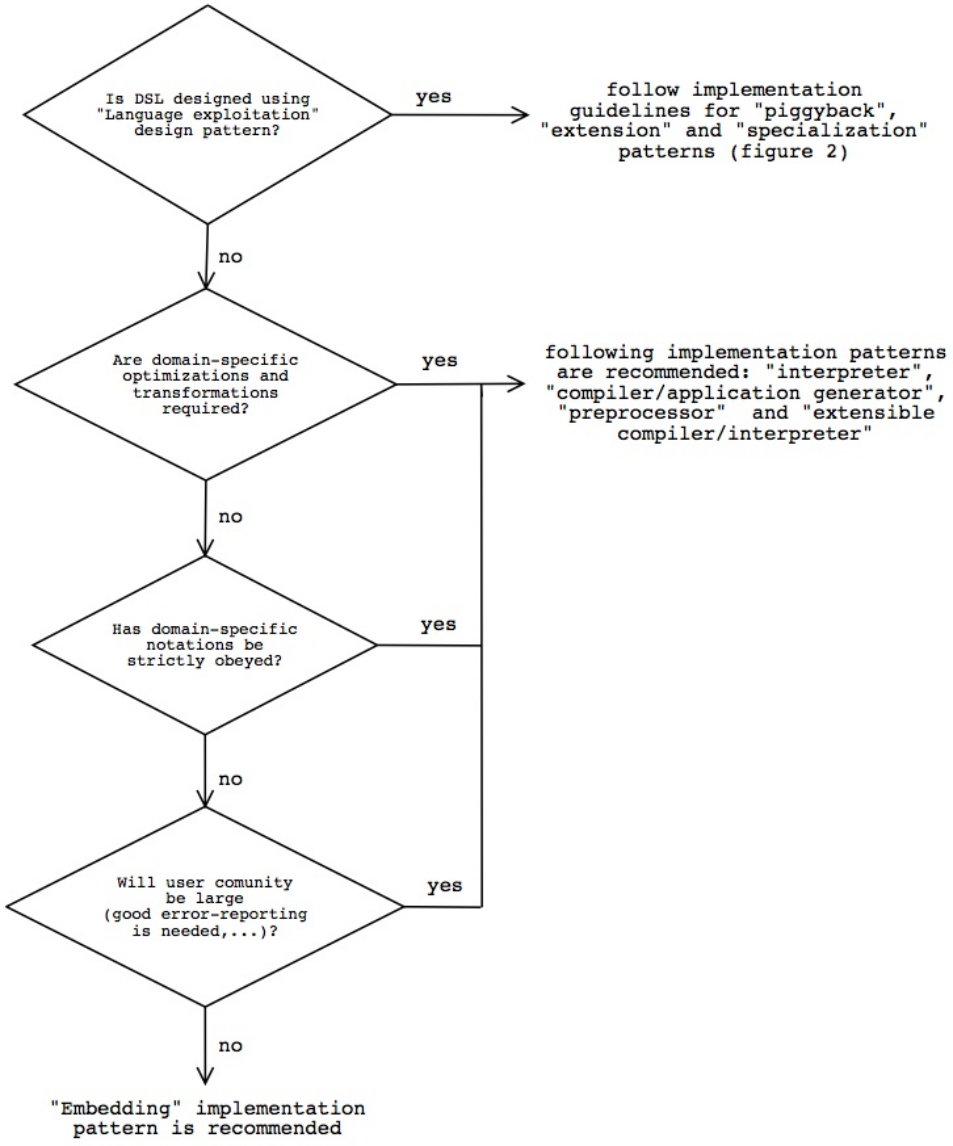


Figure 3.2: Implementation guidelines [13]

3.4 Game Development

We have decided to use an iterative development method for developing our DSL and engine. We start with an initial plan and some requirements, which then is analyzed before being implemented. It will then be tested and evaluated against our requirements. This continues until we have fulfilled all the requirements set, which leads to deployment.

We chose to use this method in order to make functional prototypes during each iteration, where we can test both the expressiveness of the DSL and its engine. This is also in accordance with Table 2.4 where we have identified our game type as "continuous". Figure 3.3 shows how an iterative development method behave.

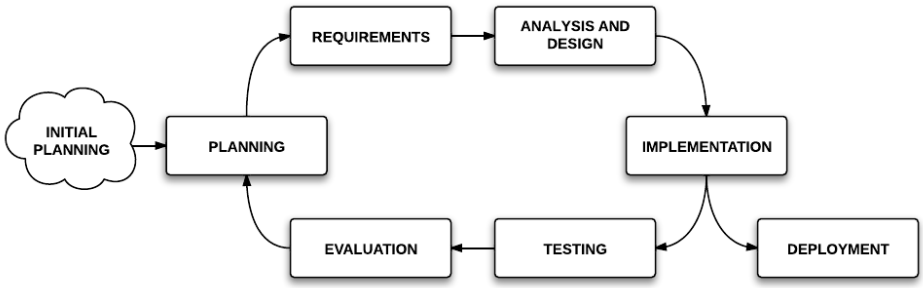


Figure 3.3: Iterative development method

3.5 Project Method

In order to create our DSL and engine we had to combine and modify the two methods described in section 3.3 and 3.4. This resulted into the iterative development method visible in Figure 3.4. The three first steps, game definition, game concepts and game specification lays the foundation for which the DSL and engine will be built on. The DSL and engine is built and expands during each iteration. After the implementation step we evaluate if the DSL and engine fulfills the requirements that we have set, see section 4.3. If the DSL and engine fulfills the requirements they are prepared for deployment. In the other case, further work is necessary and a new iteration round begins.

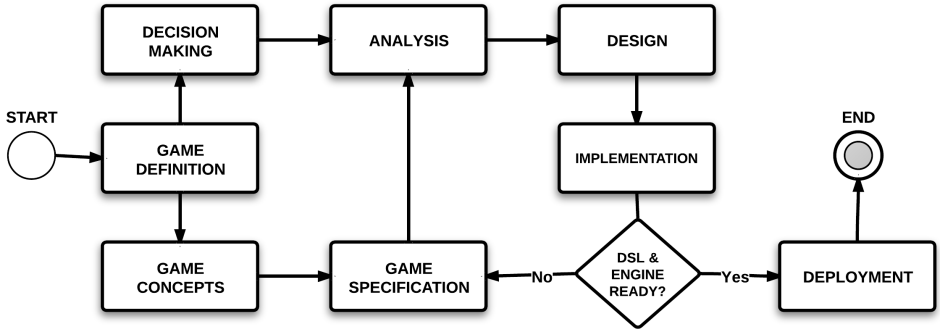


Figure 3.4: Our customized project method

3.5.1 Game Definition

This is the first step towards creating a DSL for pervasive games. It addresses the game definition, which determines what the DSL needs to sustain. The DSL made during this research is aimed to make treasure hunt games. This means we need to know what this implies. In order for a game to be a treasure hunt game, there are certain elements that needs to be present. In addition to this, we need to know what it implies that a game is pervasive.

3.5.2 Game Concepts

Based on the requirements and limitations from the previous step, a number of game concepts are designed. A typical concept has a textual description of the game mechanics and a complementary behavioral model. To avoid spending too much time making game concepts, eight concepts should be sufficient. An example of the textual description a long with its behavioral model can be seen in chapter 4 and the game concepts can be seen in appendix B.

Step	Input	Output
Game Definition	Research question and brainstorming	Game requirements and limitations
Game Concepts	Game requirements and limitations	Game concepts
Game Specification	Game concepts, preliminary study	Game elements and patterns
Decision Making	Research question, game requirements and limitations	DSL decisions
Analysis	DSL decisions, game specification	Domain definition, domain-specific terminology, domain-specific semantics
Design	Domain definition, domain-specific terminology, domain-specific semantics	DSL design
Implementation	DSL design	DSL and engine
Deployment	DSL and engine	A ready-made DSL and engine

Table 3.2: Input and output overview for each step in our project method

3.5.3 Game Specification

During this stage the game concepts are analyzed and broken down into smaller elements. This process helps to identify patterns and the elements that are frequently used to construct treasure hunt games. By undergoing a selective process the identified patterns and elements are chosen to be included in the DSL.

3.5.4 Decision Making

This step determines the approach for making the DSL. There are basically three alternatives; create a new domain specific language (DSL), reuse an existing DSL, or use a generic programming language (GPL). During this step we decided to create a new DSL.

3.5.5 Analysis

During this step it is necessary to identify the problem domain and to gather relevant knowledge for further development. The information provided by the game specification is too large to be implement at once. In order to overcome this we have to priorities the specifications during each iteration. This means new

elements and patterns will have to be analyzed in order to be implemented in the existing DSL version.

3.5.6 Design

We have decided to use *Xtext*. This means we have most of the design decisions already determined, since it covers aspects like language infrastructure, parser, compilers and more. During each iteration, we have to extend the language we have made during earlier iterations. Note that, this phase (Design) only contains the design of the DSL, more specifically, it entails grammar definition in *Xtext*, which when it is defined, *Xtext* can produce all the necessary tools, such as parser in order to interpret it. But at this stage the DSL terminology/language does not contain the domain specific semantics, that we would like. To do so, we need to implement an engine, that can extend and limit this basic *Xtext* parser to the domain in question.

3.5.7 Implementation

This phase contains the actual realization of the design phase, namely the implementation of the engine. The engine extends the basic *Xtext* functionalities, such as interpret a given DSL script (also called game model or game instance), according to the grammar definition with the domain specific semantics.

3.5.8 Deployment

The DSL and engine is released.

3.6 Toolbox

This section presents the tools and technologies we used to create the artifacts.

3.6.1 Eclipse IDE

In order to write code efficiently it is recommended to use an integrated developer environment (IDE), since they offer a source code editor, build automation, and a debugger. We ended up using the *Eclipse IDE*, which is plug-in based and support several programming languages, compilers, and interpreters. We chose to use this IDE for three reason: (1) it is free and open-source, (2) we both were familiar with it from previous projects, and (3) it got the necessary features we need for this project[3].

3.6.2 Xtext

We chose to use *Xtext* to create our DSL mainly due to the fact that it cover all the aspect of a complete language infrastructure, such as parsers, over linker, compiler and interpreter. But the most important feature is the fact that it got support for EMF, thus allowing us to load and run the instances dynamically at runtime as described in section 4.3.

3.6.3 Other Tools

The following tools and libraries were used besides the native Java libraries:

jScience is a Java tool and library for scientific calculations and visualizations. It creates synergy between all sciences, e.g math, physics, sociology, biology, astronomy, economics. It does this by integrating them into a single architecture. We have used it for real positioning[9].

swingX contains extensions to the Swing GUI toolkit, including new and enhanced components that provide functionality commonly required by rich client applications. We used this component in our engine GUI for visualizing the game world (map)[23].

OpenStreetMap is a collaborative project to create a free editable map of the world. We use this as our map provider for our engine GUI[16].

Chapter 4

Results

This chapter present our results. It starts by presenting our findings concerning the domain, before presenting the game specification. Then finally we present the system architecture from a technical point of view.

4.1 Domain

Figure 4.1 shows how the game specification was elaborated. It all starts with preliminary studies about games, which gradually focus more to our domain. The black thick lines between the boxes indicates stages where we filtered and analyzed the relevant information. The results from the analysis can be seen in the succeeding sections.

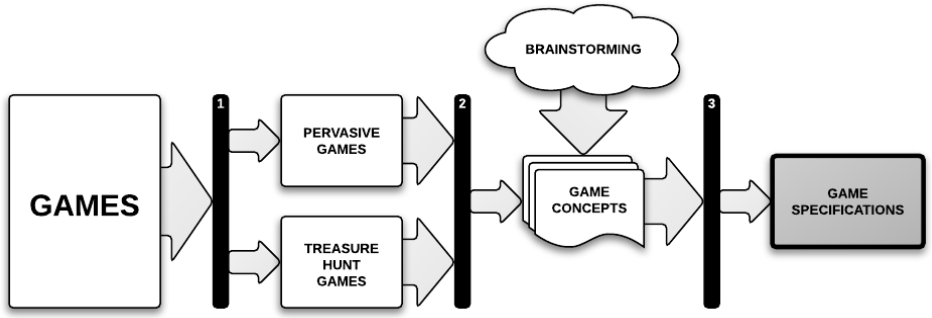


Figure 4.1: Elaboration of the game specifications

1. During this stage we gathered all the relevant information concerning games, see section 2.1. We were able to identify some key elements that are necessary for all games.

2. See section 2.1.4 and 3.1.2 for the information we found relevant. This helped us to identify some of the challenges with pervasive games, and a definition for what a treasure hunt game is.

3. By this point we have made game concepts that we will have to break down in order to find repeated patterns and game elements. Go to section 4.2 for more information about how this was done.

4.1.1 Pervasive Games

Figure 4.2 shows three elements that we find necessary in a game. The physical components defines the play area and the artifacts that are necessary for playing a game. The second element is the people that plays the game. The third and final element is the game rules, also known as the game mechanics. We will have a deeper look on how the status is for these elements for pervasive games today:

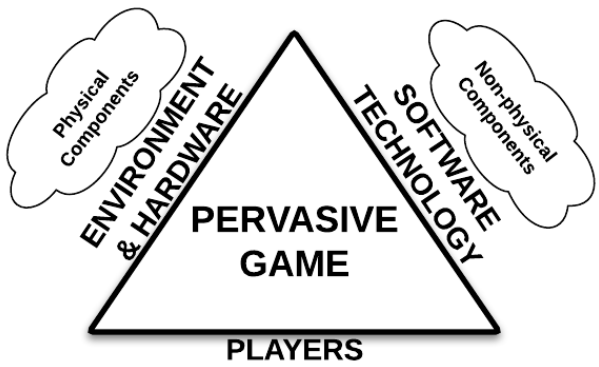


Figure 4.2: Main elements for pervasive games

Physical components: The physical world is always present, and considering pervasive games are played in the real world means we can play them almost everywhere. In addition, the world got ubiquitous technology in form of GSM masts, satellites that orbits it, and smart phones. All these artifacts can be used for supporting pervasive games.

Players: There exists many potential players in the world. The challenge is to convince them to start playing.

Non-physical components: As mentioned in the introduction of this thesis, monitoring players are difficult in pervasive games. There are very few technological solutions for this today. The pervasive games today relies on trustful players that obeys the rules or game mechanics that avoids the need for strict monitoring. This element has room for improvement, and we want to solve this by creating software.

Pervasive games faces the same design challenges as conventional games, see section 2.2.1, but they also introduce new ones. In order to identify these challenges we created a *pervasive matrix*, see Table 4.1. The matrix uses the pervasive perspectives identified in the *TeMPS* framework, and uses them as "functions" and "variables". The vertical axis on Table 4.1 represent the functions while the horizontal axis represents the variables. The result can be seen below. These challenges were used for design considerations when we designed the game concepts, see appendix B.

	Temporal	**Mobile**	**Perceptual**	**Social**
Temporal	-	T(m)	T(p)	T(s)
Mobile	M(t)	-	M(p)	M(s)
Perceptual	P(t)	P(m)	-	P(s)
Social	S(t)	S(m)	S(p)	-

Table 4.1: Pervasive matrix

T(m) *Real time.* The time depends on global position, e.g. the time in central europe is 1 hour in advance of Great Britain. A challenge that doesn't have much impact on pervasive games, but rather on online games that have players from all over the world.

T(p) *Relative time.* People perceive time differently depending on their state of mind. People usually feel that the time goes faster when they are having fun. Making an entertaining game that makes the player lose track of time should be a goal for all game developers. When this is achieved it is usually a sign that the game is a success.

T(s) *Scheduling.* People have different commitments and might not be available all the time. This is specially an issue that is real for pervasive games since they are interwoven into our daily lives. This is not a problem for conventional games since the players usually play on their own terms.

M(t) *Availability.* Some places are not available for the public continuously, e.g. stores during Sundays, or a road is closed due to construction or maintenance. This is unique for pervasive games since they use the real world as play area, which means that the constraints that exists in the real world will also affect the game world.

M(p) *Knowledge.* Objects and places can be perceived differently by people depending on their background, e.g. some people might think that Trondheim is a big city, while others might think it's small. Universal challenge for both conventional and pervasive games.

M(s) *Capacity.* Places have different capacity for handling big crowds of people, e.g. the difference between a small room and a public park. Same as M(t) in the sense that the real world puts constraints on the game world.

P(t) *Alteration.* Objects can be perceived different at different times, e.g. day and night, seasons during a year or the bystanders walking by. This is represented in both conventional and pervasive games. In conventional games there could be bugs that are fixed, or new content is implemented. The difference is that a pervasive game is more dynamic and can change very fast continuously.

P(m) *Location.* Objects can be perceived different at different places, e.g. playing a pervasive game in Trondheim will be different from playing in New York. This is unique for pervasive games that have no limitations for where the game can be played.

P(s) *Culture.* People perceive gestures differently, e.g. handshaking and bowing. A challenge all game developers face.

S(t) *State of mind.* People's behavior varies depending on time and situation, e.g. morning grumpiness. This might seem to be a challenge that is represented for both conventional and pervasive games. However, in accordance with T(s) a conventional game is played on the player's own terms, but not for some pervasive games.

S(m) *Roles.* People behave different at different locations, e.g. the same person can be a student, brother, colleague depending where he is and the environment he is in. The normal thing is that when a player starts playing he/she regards himself as a player. However, a pervasive game that is played continuously a player might experience that the roles collide.

S(p) *Opinions.* People can have different opinions and perceive things differently from others. *"A subjective belief based on results of emotion or interpretation of facts"*. This is a universal challenge.

The purpose with the pervasive matrix is that it can help to reduce problems that could occur later in development or after deployment. Examples of such problems are that the game designer adds a post that is unreachable for the players. This could be avoided by implementing a smart mechanic in the game engine that notice where the designer adds a post and sends a notification back to the designer if the location isn't suitable. Examples to this could be stores or shopping malls that aren't open all the time, a dangerous river, rooftops, or in the sea.

Regardless of their importance, features in the DSL and engine to overcome these challenges were not prioritized since it would be too time consuming and complicated to implement. However, they were used while making the game concepts. We believe it is crucial that these features are implemented later, to prevent game designers from designing non-playable games.

4.1.2 Treasure Hunt Concepts

In order to identify what a treasure hunt game is, we had to create a range of different game concepts. The game concepts were used to extract concrete game elements that would be included in our domain-specific language. We started by

reading about existing games in order to understand what it implied to be a treasure hunt game. What we learned from this we created a template to be followed and the result can be seen in Table 4.2.

Most of the fields in the template are derived from the *TeMPS* framework and Caillois's game play definitions described in section 2.1.3. Next step was to be creative and make simple game concepts by filling the template. All the game concepts that we created can be found in appendix B.

TEMPLATE FOR TREASURE HUNT GAMES	
ID:	A name and/or number to identify this game concept
Game pattern:	Treasure hunt game
Description	
A short explanatory description of the game	
Game mechanics	
Temporal:	Determines the time aspect of the game, e.g. when you play and for how long
Social:	Human entities that are required in order to play the game
Mobile:	Where is the game played?
Device(s):	Are there any devices required to play the game
Objective(s):	What is the purpose of playing the game?
Initial:	Explains how the game is started
End:	Explains how the game ends
How to win:	Tell how you can win this game and what determines the winner
How to lose:	What determines the looser in the game
Other:	Are there other rules necessary for the game to work as intended?
Game play	
Agon:	Are you competing against others?
Alea:	Are there any random aspects in the game?
Ilinx:	Are there any physical activities in the game?
Mimicry:	Does the players, or game, have to mimic anything?
Paida - Ludus:	Describe the level of structure (structured - non-structured)
Concept Art/Models	
(Model describing game behavior and the pervasive triangle)	

Table 4.2: Template for creating game concepts

4.2 Game Specification

From the game concepts we created we were able to extract game elements and patterns we wanted for our DSL. We decided to divide the patterns into the four dimensions, *goal*, *mobility*, *social*, and *temporal*. Three of the dimensions are taken from the *TeMPS framework*, but we decided to disregard the perceptual perspective due to high complexity. However, we believe that the perceptual perspective can still be addressed depending how the game designer reflects the real world in other game elements.

Furthermore, we introduce the three dimension levels, low, medium, and high, which represents the importance and influence of each dimension. The levels can be compared with Roger Caillois's definition on types of play, see section 2.1.3. *Ludus* represents the low level, since this gives the players the freedom to decide what order and when they want to find the posts. *Paida* represents the high level, since this puts more limitations on the players, making the game more structured. The dimensions and levels can be seen in Table 4.3.

	Low	**Medium**	**High**
Goal	*Find*, the player receives points by finding a post	*Find and solve*, the player get points from finding the post and solve its task	*Solve*, the player receive points by solving a post task
Mobility	There are no order in which the posts have to be found	There exists posts that can only be found when certain other posts are found	All posts need to be found in order
Social	The game is played by one person and there are no other players visible in the game	The game is played alone, but you compete against other players	The player has to cooperate with other players
Temporal	There are no deadlines for the posts to be found	Some of the posts needs to be found within a deadline	All posts need to be found within a deadline

Table 4.3: Pervasive dimensions and levels

The succeeding chapters starts by presenting the identified elements and patterns, before explaining the four dimensions and their levels more into detail. It continues by presenting the expected expressiveness of the DSL, issues concerning overlapping dimensions, and finally an iteration plan concerning the implementations of the dimensions and their levels.

4.2.1 Identified Game Elements and Patterns

Table 4.4 shows the game elements and patterns we identified from the game concepts. The elements can be categorized using the three main game elements; physical, non-physical, and player. All these elements are used for making pervasive treasure hunt games. Figure 4.14 shows a class diagram for these elements and patterns and illustrates how they are interconnected.

Physical components are represented as *world*, *post*, *position*, *task*, and *distance*.

Non-physical components are represented by the patterns which we have determined by the four dimensions goal, mobility, social, and temporal. In addition, we got the elements *level*, *order*, *clue* and *deadline*.

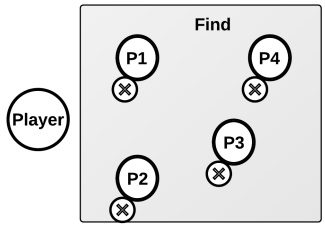
Player is represented by the element *player* and *team*.

Game elements	
World	represents the play area, which is the real world and contains all the game elements.
Player	plays the game.
Post	is located in the real world, which the player needs to find.
Clue	gives information regarding a post's location.
Task	is something the player needs to complete in order to proceed in the game.
Position	indicates the real location of players and posts in the real world.
Distance	indicates the distance from the player to the closest post.
Level	indicates the level for a pattern. LOW, MEDIUM, and HIGH
Team	is cooperative play between players.
Deadline	makes a post unreachable when the time is up.
Order	meaning that certain posts needs to be found before others.
Game patterns	
Goal	indicate how players receive points, see section 4.2.2.
Mobility	indicates the importance of post sequentiality in a game, see section 4.2.3.
Temporal	indicates the importance of post deadlines in a game, see section 4.2.4.
Social	indicates the importance of cooperation and competition in a game, see section 4.2.5.

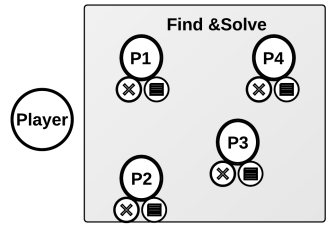
Table 4.4: Identified game elements and patterns

4.2.2 Goal

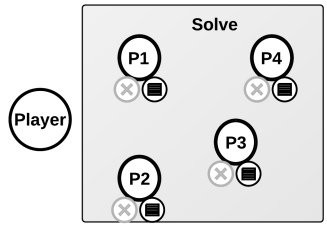
This dimension introduce some of the competitive aspects of a game by deciding how players receive points when they are playing. What the points are used for is up to the game designer, but player ranking would be an option. See Figures 4.3(a), 4.3(b) and 4.3(c) and Table 4.5 for further explanation.



(a) LOW. The task it self is to find the posts.



(b) MEDIUM. Player get points both for finding a post and for solving the associated task.



(c) HIGH. Player only get points for solving the associated task.

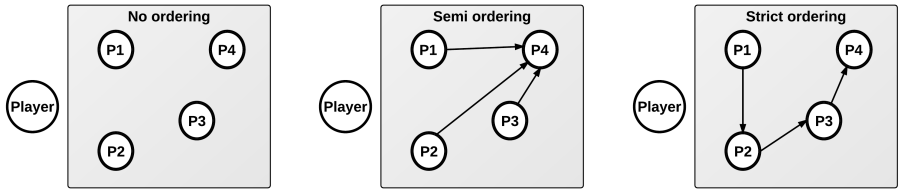
Figure 4.3: Goal dimension levels

Level	Explanation
Low	goal level means that players get points by finding a post. There are no tasks associated with the posts.
Medium	goal level means that players get points both for finding a post and for solving the task associated with the post.
High	goal level means that a player get points for solving a task associated with a post. This makes it more challenging since the player receives no points for finding the post itself.

Table 4.5: Levels affecting the goal dimension

4.2.3 Mobility

This dimension determines player movement. The game designer decides the order of which the posts needs to be found. The player needs to follow this order to be able to finish the game, see Figures 4.4(a), 4.4(b) and 4.4(c) and Table 4.6 for more information.



(a) LOW. Model showing that the player is free to find any post

(b) MEDIUM. Model showing that the player has to finish at post 4

(c) HIGH. Model showing that the player has to visit the posts in a strict sequence

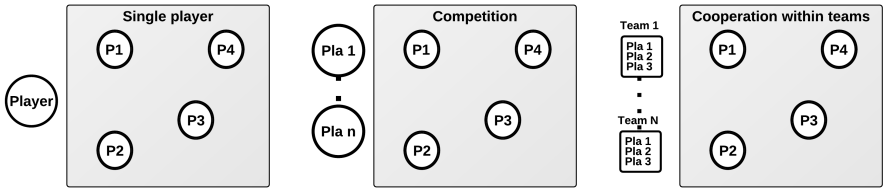
Figure 4.4: Mobility dimension levels

Level	Explanation
Low	mobility level is illustrated in Figure 4.4(a), where a player can find posts in any order desired.
Medium	mobility level means that posts are semi-ordered. If we have the four posts p1, p2, p3 and p4, the developer can specify the order for how some of these posts needs to be found. Figure 4.4(b), shows such a semi-ordering. In this scenario, a player can only find post p4 when he has found all the other posts. This can be achieved by writing the following code: `p1->p4, p2->p4, p3->p4`
High	mobility level means that all posts have a strict ordering, which means the posts is found in sequential order. Figure 4.4(c), shows and example of such sequential ordering. This can be achieved by writing the following code: `p1->p2, p2->p3, p3->p4`

Table 4.6: Levels affecting the mobility dimension

4.2.4 Social

This dimension adds competition and cooperative play into the game. Furthermore, it decides how the game handles multiple players, see Figures 4.5(a), 4.5(b) and 4.5(c) and Table 4.7 for more information.



(a) LOW. Model showing that there is only one player

(b) MEDIUM. Model showing that that the players are competing against each other

(c) HIGH. Model showing that the players are cooperating in group and compete against other groups

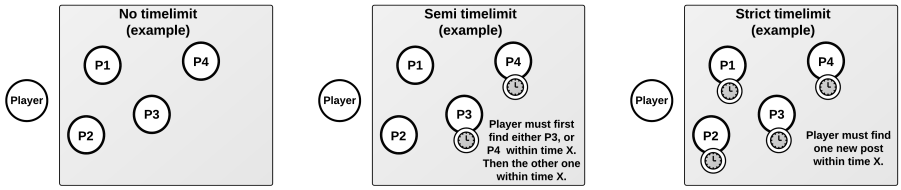
Figure 4.5: Social dimension levels

Level	Explanation
Low	social level is illustrated in Figure 4.5(a), where a player can complete the game without any cooperation or interaction with other player.
Medium	social level means that players are competing against each other indirectly, meaning no cooperation or interaction while playing. Figure 4.5(b), illustrated such an instantiation.
High	social level means that the players are in teams and they have to complete the game with each other while competing against other teams. If we have four players pl1, pl2, pl3 and pl4, the developer can specify the teams by the following code: `teamA(pl1,pl2),teamB(pl3, pl4)`

Table 4.7: Levels affecting the social dimension

4.2.5 Temporal

This dimension adds time constraints in the game, see Figures 4.6(a), 4.6(b) and 4.6(c) and Table 4.8.



(a) LOW. Model showing that there are no time limits to visit the posts

(b) MEDIUM. Model showing that some posts have a time limit

(c) HIGH. Model showing that each post have a time limit

Figure 4.6: Temporal dimension levels

Level	Explanation
Low	temporal level is illustrated in Figure 4.6(a), where a player can find a post without any time constraints. This means that the game can be played as long as the player wants.

Medium	temporal level means that one or more posts can have a time constraints. For instance, if we have posts (p1, p2, p3 and p4), the developer can specify time constraint this way:
	`deadline: 0d 0h 5m 0s // 5 minutes`
	`posts: p3, p4`
	Figure 4.6(b), shows an instantiation of such a semi timeout. In this scenario the player needs to find post p3 or p4 within 5 minutes, else the game ends and the player will lose the game.

High	temporal level means that all posts have a time constraint. Figure 4.6(c), shows a game with such strict time constraints. In this scenario a player has to find a new post every 5 minutes, else the player will lose the game.

Table 4.8: Levels affecting the temporal dimension

4.2.6 Overlapping Dimensions

The dimensions and their levels introduce some overlapping design conflicts that needs to be addressed. By conflicts we mean that the dimensions will affect each other depending on their levels. Note that the goal dimension is not included since this only decides how the players receive points, thus no conflicts occur.

D1/D2	LOW	MEDIUM	HIGH
LOW	None	None	None
MEDIUM	None	Medium	High
HIGH	None	High	High

Table 4.9: Dimensions affecting each other

There are nine situations where conflicts occur between two dimensions, see Table 4.9. Each type of overlap will affect the involving two dimensions with different magnitude. We have decided to categorize these magnitudes into three levels, none, medium, and high. None, indicates there are no actions necessary to overcome the conflict. This solves itself by letting the dimension with the highest level dominate over the other. But for the two other magnitudes it varies depending on what the dimensions are.

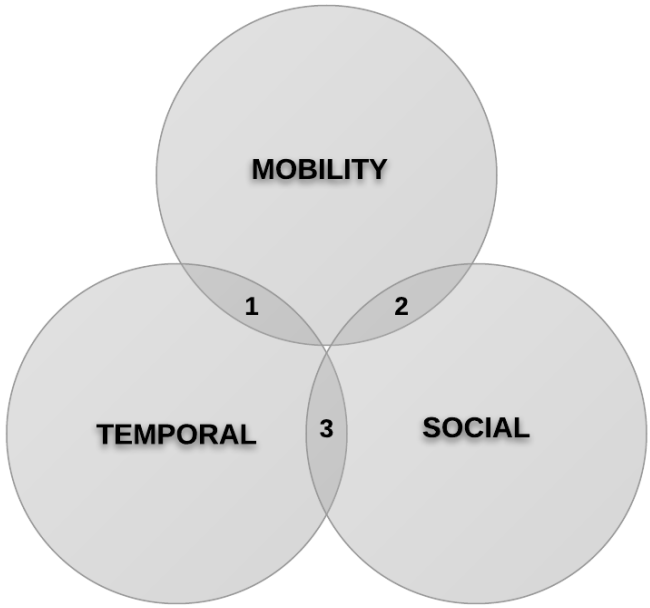


Figure 4.7: Overlapping dimensions

1. **Temporal vs Mobility** The typical problem in these cases are that the player won't have enough time to reach their posts since they have to find

the posts in some sort of order. It is therefore important that the deadlines and ordering are coordinated so that the players are able to finish the game.

2. **Mobility vs Social** This problem was identified in section 4.1.1 and is equivalent with *capacity* and *accessibility*. The game should be designed so that the game is equal for all that plays it. This means that some of the post locations will have to be able to support multiple players simultaneously, and should be accessible at all times. This means that the game designer needs to be consequent with where he places the posts in the real world.

3. **Social vs Temporal** is equivalent with *mobility vs social*.

4.2.7 Iteration Plan

In order to avoid spending too many resources in the implementation phase we couldn't implement all the dimension levels at once. We felt it was important to spend less time programming so that we could get feedback from testing and evaluating the DSL and engine by creating games. During each iteration we improved and/or added more features.

Initial Iteration

The very first DSL and engine, did not have the defined dimensions and all the elements implemented. This prototype was mainly made for discovering the capabilities and limitations with the *Xtext* framework. However, the necessary elements like *player* and *post* were implemented. In order to test the games made with the DSL we needed to develop a game interface, see appendix C for more information. Figure 4.8 shows how the pervasive dimensions appears in the first prototype.

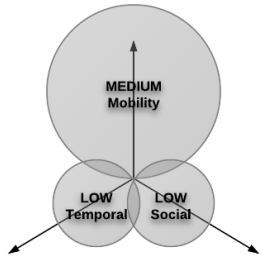


Figure 4.8: First prototype

We redefined the DSL grammar slightly by implementing the goal dimension in the second prototype. We also chose to improve the game interface by implementing a real map to the interface using *openstreetmap*, see appendix D (called strategy).

Main Iteration

Figure 4.9 show the third prototype that was implemented. This is where we implemented the three dimensions and levels. This made it possible to add post ordering, post deadlines, and teams to some degree. This was the iteration with the biggest workload. More information can be found in appendix E.

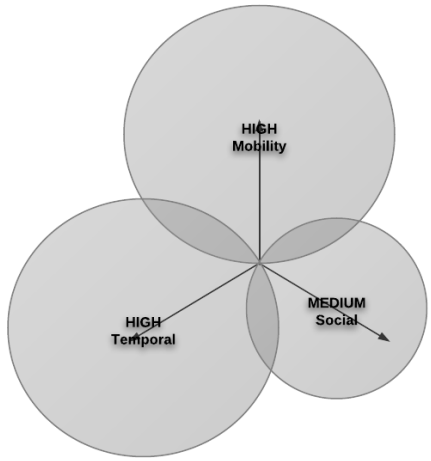


Figure 4.9: Third prototype

Final Iteration

The forth and final prototype doesn't add much more to the DSL grammar. However, we wanted to address usability by making the game interface (prototype) into an editor as well. This means the creator doesn't have to write code themselves since this will be handled by the editor. More information can be found in appendix F.

4.3 System Architecture

A system architecture is a conceptual model that defines the structure and behavior of a system. This is done by presenting different views. These architectural views, as described in [5], is made in order to depict different aspects of the system (DSL, engine and editor). It is intended to capture and convey the significant architectural decisions used to build our system. The different views are meant to give meaningful information about the architecture for one or more stakeholders. Before we introduce the architectural views we start by introducing the stakeholders and functional requirements for this project. It continues by introducing some technical constraints before describing the overall system design.

4.3.1 Stakeholders

Stakeholders are people, organizations, or objects that are directly or indirectly involved with our project. They can either affect or be affected by our project or system, which means they are important to identify. The direct stakeholders are identified in Table 4.10.

Stakeholder	Rationale
Players	Are the ones that play the games created with our system.
Domain experts	Are experts within pervasive games, and/or treasure hunt games.
Developers	Everyone that wants to create a treasure hunt game using our system.
NTNU	Project owner.

Table 4.10: Stakeholders

4.3.2 Functional Requirements

A functional requirement is associated with a specific function, task or behavior, which the system has to support. There are some key requirements that have a significant bearing on the architecture, which also provide a baseline for validation and verification of the deliverables. The most important functional requirements identified for our system can be seen in Table 4.11. These requirements were elaborated during the game specification phase, and were modified or added during the iterations through the development of the system.

Req. ID	Artifact	Description
FR01	**DSL**	The DSL (.th) should provide (have) textual representation.
FR02	**DSL**	The DSL (.th) should be readable by a domain expert.
FR03	**DSL**	The DSL (.th) should be writable by a domain expert.
FR04	**DSL**	The DSL (.th) should support all the pervasive dimensions.
FR05	**Engine**	The engine should provide import and export functionalities.
FR06	**Engine**	The engine should interpret a game model of Treasure hunt DSL (.th)
FR07	**Engine**	The engine should interpret the game model created in .th DSL such as the pervasive dimensions according to the specified semantics.
FR08	**Engine**	The engine should provide start and stop functionalities.
FR09	**Editor**	The editor (textual) should support domain-specific validation.
FR10	**Editor**	The system should provide an easy to use graphical editor for creating game models
FR11	**UI**	The engine should provide a map view for admin user in order to monitor players.
FR12	**Engine**	The engine should provide a function for admins to edit players/posts and dimension configuration info dynamically (in runtime).

Table 4.11: Functional Requirements

4.3.3 Technical Constraints

A constraint is an element factor that works as a bottleneck for an entity, project, or a system. They put restrictions on the system from achieving its potential, which makes them important to identify. Table 4.12, shows the technical constraints on the platforms we have chosen to work with.

Technical Platform	Description
Server side	The engine is hosted on an *Apache* web server. The web server is listening on the web standard port 80. Web server will accept all requests from the clients and forward them to the specific engine server hosting location. All communication with clients have to comply with public HTTP, TCP/IP communication protocol standards.
Client Side	The clients will be able to access the engine only through the web (www), which implies that they need access to the Internet (Wi-Fi or 3G). Clients are required to use built-in technology like GPS, which are present in most smart phones today. The targeted operating systems for the clients are *Windows phone*, *iOS* and *Android*, which implies we need cross-platform support.
Xtext	The *Xtext* framework can only be used with the *Eclipse IDE*.

Table 4.12: Technical Platform

4.3.4 Overall System Design

Figure 4.10, shows a simple process of how our system works. First step is to create game models using our DSL, which has the file extension *.th*. The model is then passed to the engine in order to be interpreted. Examples of a models written in *.th* can be seen in appendix E. As for the graphical editor, we have integrated it with the engine. By doing this we could eliminate the need for a text-editor and an IDE for model creation, which simplifies the creation of a model. For more details regarding the graphical editor see appendix F.

The engine will be implemented with a client-server pattern when interpreting a game model (.th), see Figure 4.11. The players communicate with the engine and server through the HTTP protocol with their clients (smart phones). The engine will provide the clients with the necessary information about their own state (or view) in the game, and not the state of other players. By doing this, a player is not able to get information they shouldn't have access to, thus avoiding the possibility for players to cheat in the game.

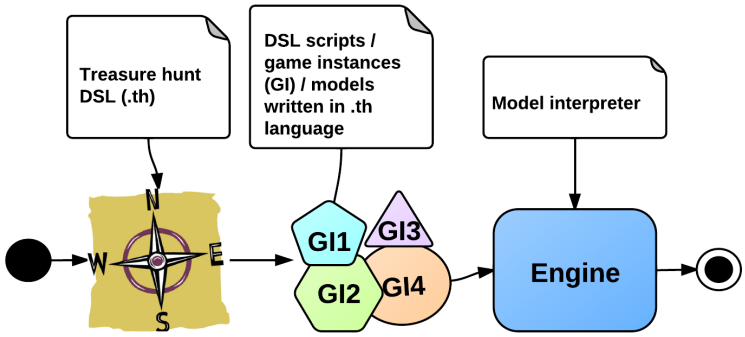


Figure 4.10: Code generator strategy vs Engine strategy

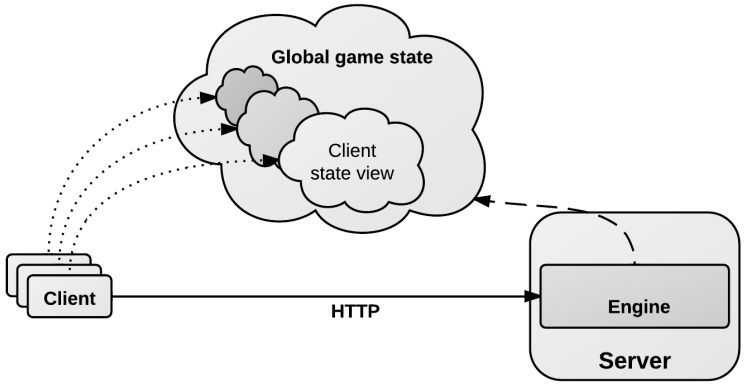


Figure 4.11: Engine - Client-Server pattern

4.3.5 Architectural Views

The views we have chosen to use can be seen in Table 4.13, which also shows who they are intended for.

View	Audience	Area	Related Artifacts
Use case	All stakeholders of the system.	Describes the set of scenarios and/or use cases that represent some significant and central functionality of the system.	Use-Case diagram
Implementation	Programmers.	Software components: describes the layers and subsystems of the application.	UML class and package diagram
Process	Programmers and integrators.	Non-functional requirements: explain the system processes and how they communicate. Document the runtime behavior of the system.	UML sequence and activity diagram

Table 4.13: Separation of Concerns (views)

Use-Case View

We made a total of five scenarios that describes situations where stakeholders uses different functionalities provided by our system. These scenarios played a significant importance on the architecture. Figure 4.12 shows the most important use-cases that helped to form the system architecture.

1. **Play games**. A player plays a treasure hunt game using a smart-phone with an installed application (client). The player navigates through the real world, in Trondheim city, looking for posts. The player is initially given a clue for the location of the first post. While the player walks around the play area, notifications about his/her distance to the post are shown on their smart phones. Once the player has found the post the server validates the position and the player receives a task that needs to be completed. After the task is done, the server sends the player a new clue for the next post.

2. **Create games/models**. A domain expert/developer creates a treasure hunt game using a textual editor. He/She creates a set of posts and places them in real locations using GPS coordinates. For each post the developer has to initiate the following variables, title, the clue for next post, and the task. Then the developer has to configure the pervasive dimensions levels. The developer has now created a game, which is ready to be played.

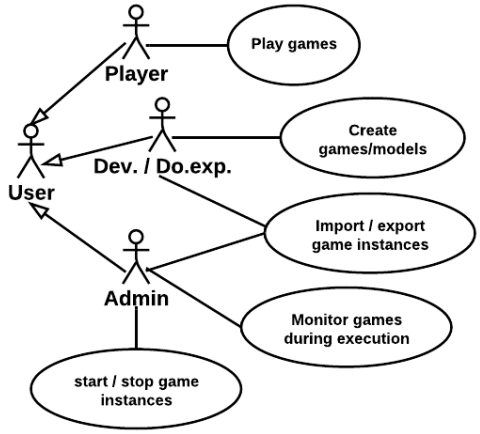


Figure 4.12: Architectural significant use-cases

3. **Import/Export game models**. A domain expert/developer selects a file with the *.th* extension to be imported into the engine, so that it can be configured, executed, and interpreted. After the configuration the game model is exported from the engine to the same file.

4. **Start/Stop game models**. A player plays a game when a break is required. The player push a pause button, which pauses the game. When the player is ready to play again he push the pause button again, which continues the game.

5. **Monitor games during the execution**. An admin can monitor the game progress while the game is being played using an engine GUI. It provides a map where posts and players position are visualized and updated in real time. The admin selects a post by clicking on it, and a detailed view of that post is displayed. The same happens when the admin selects a player. The admin see that the engine GUI provides all the default administrative capabilities, such as throwing a player out of a game, adding new posts, or altering existing posts.

Implementation View

Figure 4.13, shows the main packages involved in the graphical editor, engine and its GUI.

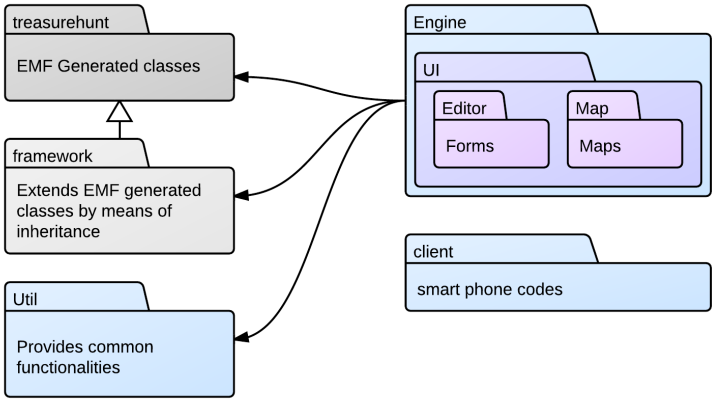


Figure 4.13: Package and subsystem

1. **treasurehunt package** includes all the EMF generated classes by *Xtext* from the DSL definition. We use these generated classes in our engine in order to load a game model and execute it dynamically at runtime. See Appendix F for more details regarding DSL definition.

2. **framework package** includes all the classes that are extensions to the EMF generated classes. By extending more functionalities to some of the EMF generated classes we are able to hide unnecessarily details from the developers using the treasure hunt DSL. Figure 4.14 shows both the EMF generated classes (treasurehunt package) and all the extended classes that inherits from the generated classes (framework package).

3. **util package** includes all the common functionalities that are required in different parts of the system.

4. **engine package** includes both the graphical editor for creating game models, the engine for interpreting them, and the GUI for admin monitoring.

5. **client package** includes client (smart phone) specific codes. However, we have not implemented this functionality in our prototype yet.

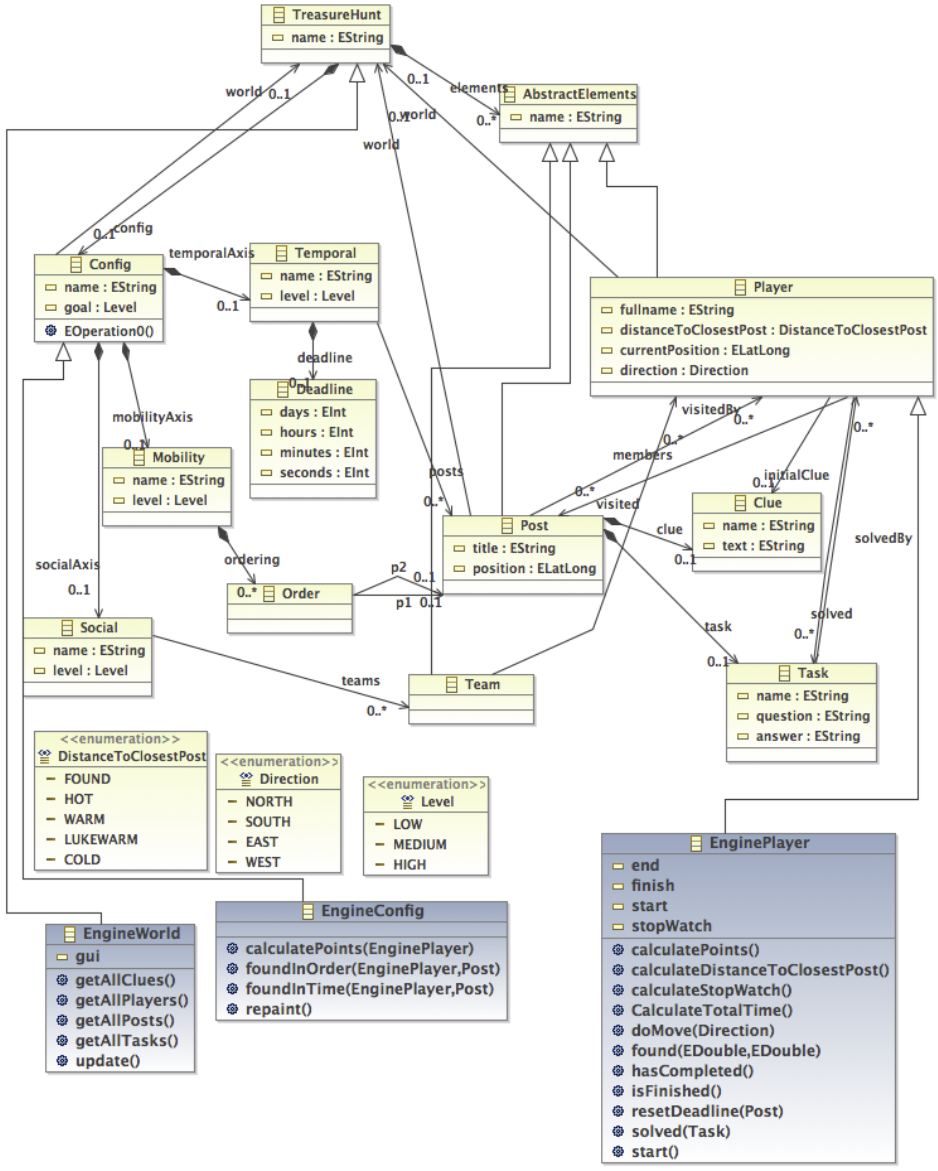


Figure 4.14: Extended EMF generated classes by *Xtext*

Process View

We have integrated the graphical editor with the engine to a single software component, which allows us to create and execute a model of a treasure hunt game in the same GUI. Figure 4.15, shows the process as an UML activity diagram. Once the application starts, we got the options to create a brand new model or to import an existing one by selecting a file with the *.th* extension. Once a game model is imported, we are then allowed to edit it.

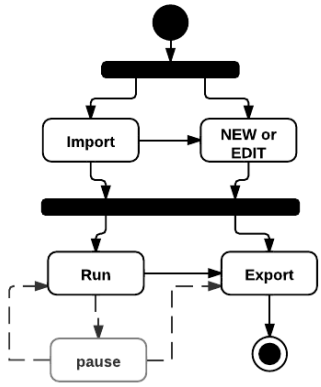


Figure 4.15: Activity diagram for creation and execution

When a game model is created we got the option to either export it as a file or to execute it. If it is executed (run state), the engine interprets the game model and waits for player actions. Figure 4.16 shows the main game loop where the engine waits for user input, which process the input then updating the global game state. This information is then sent to each client to update their individual game states.

Figure 4.17, shows the sequences of method call between clients and the server/engine. Notice that in this diagram we use a location service in order to validate that a user is actually at a particular location. The *updateState()* method updates the global game state. This method is called when a player finds a post or solves a task associated with a post.

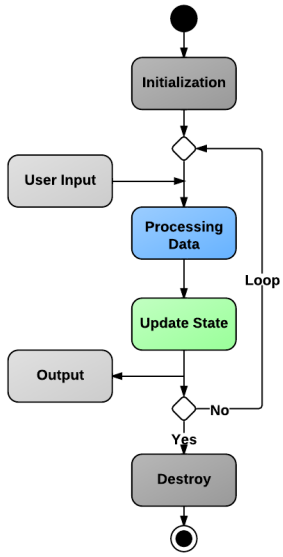


Figure 4.16: Main Game loop - runtime model

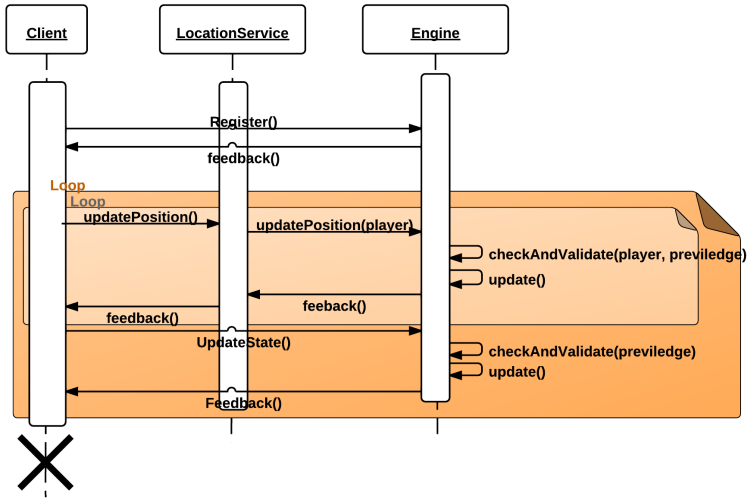


Figure 4.17: Client-Server - main game loop

Chapter 5

Discussion

This chapter present our discussion concerning both the results and the methodology used. It starts with a discussing of the project method, then moving on to our treasure hunt DSL. It also contains a discussion concerning the DSL engine and its graphical editor.

5.1 Project Methodology

5.1.1 Project Method

Due to the nature of our task, we had to invent a new project method, which we elaborated in section 3.5. We did this by combining the two well established methods; iterative game development and a DSL lifecycle.

We anticipated that the requirements would change during the project as we learned more about the domain, which they did. Therefore we chose to use an iterative development method in order to create the artifacts. It enabled us to easily add and remove requirements based on both new ideas and changes, excerpted from the testing and evaluation of the treasure hunt games we were able to make.

The project method worked well for our project. Specially, the most crucial part of the development method was the initial steps; game definition, game concepts, and game specification. This was important since this is where pervasive games were investigated. In addition, the DSL terminologies and domain specific abstractions were defined during this step, which determined the basis for the entire project.

5.1.2 DSL Development

A DSL lifecycle consists of five phases, as mentioned in section 3.3. In each phase the DSL designer has to choose one or more patterns, and we made the following decisions:

Decision We chose to develop a new DSL (.th). The main reasons for this decision were the following: (1) We had a pretty good understanding of the domain from our previous in-depth projects, [8] and [21]. Domain concepts such as post, player, task, clue, were already defined during those projects. (2) There are no DSL for pervasive games, therefore we couldn't reuse an existing DSL. (3) We wanted to develop further concepts and abstractions that are specific to the domain of pervasive games, thus using a GPL was not a real option for us.

Analysis During this phase we gathered domain knowledge mainly from the following sources: literature study on pervasive games, literature study on treasure hunt games, and making treasure hunt concepts, see appendix B. These sources were used to produce a set of domain-specific terminologies, such as the pervasive dimensions and treasure hunt elements such as *posts*, *players*, *tasks*, and *clues*.

Design We chose language invention over language exploitation, mainly because we wanted to create concepts and abstractions that are specific to the domain. This would make our DSL both readable and writable by domain experts. Furthermore, we chose a formal specification over an informal one, when specifying the design before implementation. As mentioned before in section 3.3, a formal description uses an existing semantics definition method, which in our case is extended backus-naur form (EBNF). In this phase our choices were influenced by the technology we had chosen to use, namely *Xtext*.

Implementation We chose the interpreter pattern, mainly due to: (1) We wanted the DSL syntax to be close to notations for the domain in order to be both readable and writable by a domain expert. (2) Domain-specific analysis, verification, optimization, and transformation (AVOT) possibilities. (3) Scientific curiosity.

We managed to integrate the DSL life cycle method very well into our project method. During each iteration we would do the analysis, design, implementation phase multiple times. However, we never reached to the deployment stage since our current DSL and engine are not ready to be used by the audiences.

5.2 Treasure Hunt DSL

5.2.1 Treasure Hunt Concepts

In section 4.1.2 we talked about treasure hunt game concepts and the template we had used to make them. The concepts were used to extract game elements and patterns to be included in the DSL, thus making this a critical stage in the DSL development.

The template was based on the preliminary study concerning information about all games and information about pervasive games. We used the template as a framework for making the game concepts. This made it easy for us to create concepts by filling in the required information in each field of the template. In hindsight, we think this might have added some limitations on varieties and originality in the game concepts. However, at the same time we believe this approach made it easier for us to extract the game elements and patterns we would base our DSL on. Other approaches for making concepts, like free text, would also have worked, but would require more work in analysis and breaking them down to elements and patterns.

There was a total of eight game concepts that were made during this project, four by each designer. We felt that this was an adequate number since we believed that a higher number wouldn't have made much of a difference. This would be both time consuming and repetitive. However, the number of designers could have been raised in order to absorb the ideas by individuals with different views and perception on the topic. The template could have been distributed to a number of people, which then would create their own game concepts and send them back to us. This would be a great opportunity to test and to evaluate the template itself and to provide us with original ideas for treasure hunt games. The reason why this wasn't executed was due to the huge amount of time that would be spent on management and on the analysis of the feedback. There is also a chance that the concepts they made would be very similar in the end, thus wasting our project resources.

The concepts we made can be found in appendix B. As mentioned earlier we made four concepts each without any instruction on how to do it. In addition, we had no insight in what the other had done after all concepts were made and shared. As a result, repetitiveness is noticeable in some of the concepts. One designer tried to avoid using game mechanics of other concepts resulting in four distinct concepts, while the other designer built more complexity on top of earlier concepts. We believe this could have reduced some varieties of treasure hunt games. In retrospect we should have made it more clear from the beginning that the four concepts should be distinct from each other.

In addition to the textual fields in the template we added a behavioral model. This model was made using elements from the business process model and notation (BPMN). See Figure B.1 for an example. The rational behind these models was to provide a visual representation of the game so that is was easier to see and understand the game flow. As a consequence, this gave us an idea that this could be used as a graphical notation for our DSL. The end user would create treasure

hunt games by combining game elements through a visual presentation by using a game editor. However, due to project limitations the idea never took form and remains just as an idea that could be implemented later.

The identified elements and patterns can be found in section 4.2.1 and they where the direct result from analysis made of the game concepts. This was done by comparing each game concept with each other in order to find common elements and patterns. It was a relatively fast process, since all the game concepts followed the template, and therefore easy and fast to compare. We felt that the result was good, and we believe that not much could have been changed to improve the analysis of the concepts.

To sum up, we are very satisfied with the game elements and patterns excerpted from the game concepts. The template proved to be expressive enough to design many distinct pervasive treasure hunt concepts. We also believe that our template can be used to design other pervasive games.

5.2.2 Pervasive Dimensions

In section 4.2 we introduced the four pervasive dimensions, goal, mobility, social and temporal, which describes the game patterns. The pervasive dimensions were extracted from the *TeMPS* framework, and the goal dimension from the template and the eight game concepts.

The goal dimension describes the objective in the game, implying how the player receives points during the game and how to win. We found this necessary to establish a purpose for the players in the game, so that they know how they can win and finish the game. From the game concepts we designed we excerpted three kinds of objectives which were repeated; finding the posts, solving a task, and a combination of both. We believe that three goal types were sufficient and found therefore no reason to elaborate more.

The mobility perspective was initially indented to describe the play area when we created the template. In the game concepts we see that this play area is comprised to a larger geographical area, like the university campus or the city. However, we needed a pattern for describing player movement in the game and therefore assigned the mobility dimension to describe three dimension levels of player movement; freedom, some freedom, no freedom. We felt this was a good and a sufficient solution for describing player movement in treasure hunt games, thus felt no need to expand further.

The social dimension was the most challenging to determine. Similar to the other dimensions we decided to have three pattern to describe the social aspect of the games; single player, indirect interaction, and direct interaction. We decided that the social dimension describes how players interact with each other. The challenge was that there were few examples of cooperative play in the game concepts we had made, see appendix B. As a consequence, we only got one dimension level represented in the game concepts, which includes direct interaction between players during the game. Regardless of the lack of cooperative play, we still feel we made the right decision because we believe that this dimension would be challenging to

implement in our DSL. The DSL still don't have the last two social dimension levels, indirect- and direct interaction, implemented yet for this reason.

Temporality concerned about the time aspect in the games, and we identified two types how time were used in the game concepts. The first type involved the duration of a game, how long does the player spend in order to finish the game, which would be used to rank players. Another type was the use of deadlines, which implied that the player had to find posts within a certain amount of time. The latter type was decided to be implemented in the DSL since this was easier to divide into three dimension levels; no deadline, semi-deadline, and full-deadline. Game duration is also possible to calculate, but further work is required for this to function.

We believe that the pervasive dimensions were key success factors for making the DSL (.th) as robust and expressive as it is. In addition, we didn't face much problems to implement these dimensions and their levels, besides the social dimension. We believe that this method can be used to make DSL for other pervasive games or they could build on top of our existing DSL.

5.2.3 DSL Expressiveness

The pervasive dimensions were implemented in the DSL to establish some design boundaries. The reason for doing so was to create a DSL that is easy to use, but at the same time are capable of making many different games. In addition, this decreased the complexity for making the DSL itself. We calculate that with the four dimensions and three levels it is possible to make 81 unique types of treasure hunt games. All these types can be found in Table E.2 in appendix E. This is due to the permutation rule which emerges from combining different leveled dimensions with each other:

$$f(n) = n^r$$
$$f(n) = 3^4 = 81$$

This doesn't set limitation on the maximum amount of different treasure hunt games, but there is a limit for combining the dimensions and their levels. We felt that this was sufficient since the games can be changed further by adding and/or altering the other game elements, e.g. *posts*, *clues*, *tasks* and *world*. Exactly what you can change can be seen in Table 5.1. In principe it is possible to create an unlimited number of different pervasive treasure hunt games with our DSL, but within the boundaries set by the elements and patterns themselves.

If we want to increase the number of unique types of treasure hunt games, or maybe make the DSL suitable for other games, we could raise the number of levels and dimensions. A possibility is to break down the pervasive dimensions into smaller dimensions, e.g. mobility could be post locations, play area, and player movement. If we divided one of our dimensions in half would give us a total of 5 dimensions. The permutation formula would then give us a total number of 243 unique treasure hunt games. Another possibility could be to add a forth level, e.g. low, mid-low, mid-high, high. The permutation formula gives us 256 unique games

Element	How to add expressiveness
World	Also called play area, can be set wherever in the real world. However, we recommend that the play area is set according to how the game is intended to be played. An example is games that are played by foot should have a smaller area than a game intended to be played with a car or bike.
Posts	Can be set at any location using real GPS coordinates. In addition, there is no limit for how many posts that can be added, so in theory the creator can add an infinite number of posts.
Clues	Can be added on posts using free text. This means the creator can basically decide himself how he wants the players to find posts using the clues. Examples of such clues are GPS coordinates, riddles, description, itinerary, and more.
Tasks	Is an area which needs further expansion. The DSL supports Q&A's where the creator can ask anything he like. Examples of such questions are location-based, history, general, and many more.
Teams	Can be made.
Deadlines	Can be added to the posts. This deadline can be everything from years to seconds and everything in between.
Order	Of how the posts needs to be found can be decided by the creator.

Table 5.1: How elements can expand the expressiveness

with four dimension levels. This indicates that it is possible to raise the level of abstraction by increasing the number of levels and/or dimensions. The graph in Figure 5.1 shows how much the number of unique treasure hunt games raises by adding either another level, indicated by *g(x)* with a red line, or adding another dimension, indicated by *f(x)* with a blue line.

We are satisfied with the level of expressiveness in our DSL. Considering it is meant for creating pervasive treasure hunt games, we don't see the need to expand the expressiveness by adding more dimensions or levels. This would have required more time in implementation and might have resulted in fewer iterations. In addition, we don't see how raising the expressiveness would have made it more obvious to see if a DSL is an appropriate solution for making pervasive games. However, if we were forced to expand we would choose to add another dimension, which would make a total of 5 dimensions. The reason for this is that this would require less time to implement and at the same time had the least chance for ripple effects in what we already implemented. Adding another level would have forced us to redefine all the other levels for the dimensions, which is a total of 16. In comparison, if we added another dimension we would simply divide an existing dimension, which means we would just have to add 6 new levels. We are convinced that these dimensions and levels are the correct path in making a DSL for pervasive games. Expanding them could result in a DSL that is able to create any type of game.

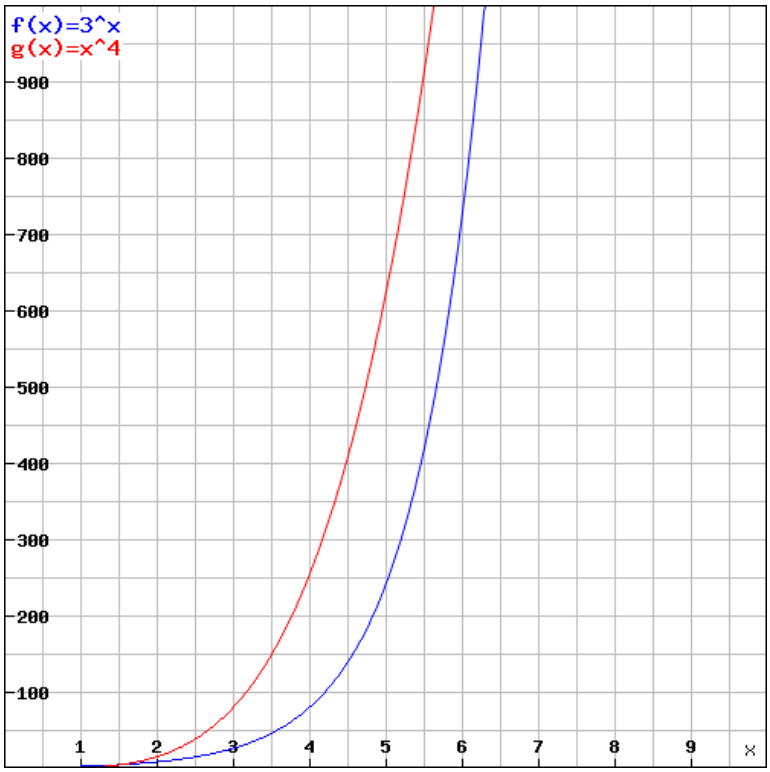


Figure 5.1: Number of unique treasure hunt games

5.2.4 DSL Outside its Purpose

In section 3.1 we ask a research question concerning the use of our DSL outside its purpose (RQ3). With the last version of our DSL we tested if we could make other games than treasure hunt games. These games were only prototyped to see if they could be made, thus never play tested.

We started by creating games that have the most similar game mechanics and dynamics, which is a scavenger hunt game. The player would receive a list of items, as an initial clue, which the player would have to find and photograph. The catch was that these items only existed in specific locations and these were indicated as posts. Typical items were certain buildings, road signs, restaurants, and stores. The player would have to walk to these posts and try to find the items which needs to be photographed. However, since we have no mechanic to give points based on the photographs taken and to crosscheck them with the list, this would have to be done manually. In addition to the scavenger hunt game, we were also able to create a racing game. This was done simply by making a track with posts and add a strict ordering so that they would have to be found in sequence. The timer would start when the first post was found and stop at the last. The last game we were able to create was an orienteering game. This was done by distributing posts that could be found in any order, and having the timer start when the first post was found, and stop at the last one.

In addition to the games above we tried to make the games mentioned in section 2.2.4. The tag-based game, *Pac-Manhattan*, couldn't be made with our DSL and engine. We believe for this to be possible we need to work more on the social dimension, since there is a lot more player interactions in a tag-based game. A possible solution to this would be to add a dimension regarding player roles, since in a tag game you are either the chaser or being chased. Concerning the *GeoQuiz* game there was only one key feature lacking in our DSL and engine, which was that the player could add new posts with a Q&A task. Besides that, the game would be fully possible to create.

We were able to prove that the DSL and engine could be used to make other games than treasure hunt games. However, these games have similar game mechanics and dynamics as treasure hunt games. In order to expand this we would have to increase the number of dimensions, levels, and/or add new game elements. Figure 5.2 illustrates the current state of our DSL and engine. It is designed for treasure hunt games, but it is also possible to create games similar in mechanics and dynamics.

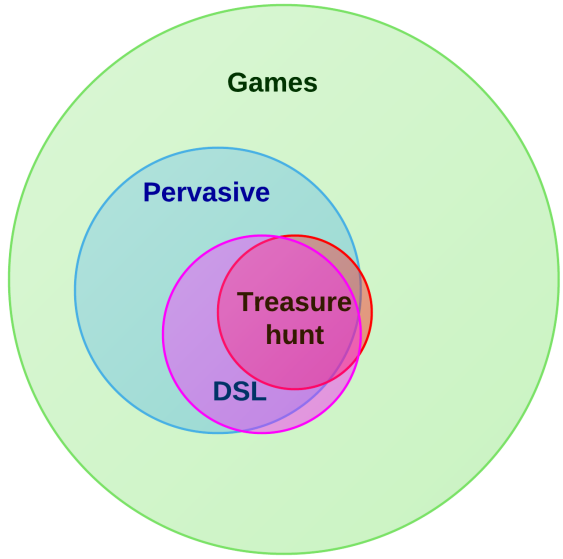


Figure 5.2: DSL coverage

5.3 DSL Engine and Graphical Editor

The treasure hunt engine and its graphical editor is simply a proof of concept. It shows that the treasure hunt DSL (.th) that we proposed is not just an idea, but given an game model of treasure hunt DSL it can actually be executed. The final engine is the result of several development iterations and most of the source code had to be rewritten/factored at least three times. These changes concerned both the grammar definition (DSL) and the underlying architecture of the engine.

In the first prototype the DSL covered most of the basic domain concepts and abstractions. The main goal was to familiarize ourselves and to discover the possibilities of *Xtext*, as mentioned in section 4.2.7. As a consequence, we only implemented the main functionalities of the engine. See appendix C for more technical details.

In the second prototype the DSL supported real GPS positioning, making the whole world a potential play ground. In order to support this functionality the engine had to be rewritten. We implemented the support for a real word map by integrating jScience[9], swingX[23] and openStreetMap[16] in the engine. See appendix D for more technical details.

During the third prototype the DSL was expanded to support the pervasive dimensions described in chapter 4. This was a big change in the DSL, which meant the engine had to be rewritten once again. During this iteration we were able to implement the engine to support all the pervasive dimensions except the social one. However, some of its functions were partially supported. See appendix E for more technical details.

In the forth and final prototype there were no changes to the DSL itself. However, in order to simplify the creation of treasure hunt games we integrated a graphical editor with the engine. By doing this we hoped to eliminated the need for an external IDE or text editor, and making it more user friendly for non-technical users. See appendix F for more technical details.

The time spent developing the engine has given us insight in the importance of creating something that executes the DSL. We experienced that it isn't the DSL itself that was challenging to create, but more the realization of it by construction an engine. It is the implementation of the engine that has consumed most resources, since this requires a lot of time writing code, and to test and evaluate the prototypes. In hindsight, we feel we could have been better prepared for the development of the engine, since a lot of code was rewritten during each iteration. By better planning we could have saved a lot of time rewriting code, and we could have spent that time focusing on the DSL instead. However, we are satisfied with the results we were able to achieve.

Chapter 6

Conclusion

This chapter presents the conclusion for this master thesis and further work.

6.1 Conclusion

The three pervasive dimensions, mobility, social, and temporal, proved to be a good starting point for generalizing pervasive game elements. We also found it necessary to have a dimension that determined how the player received points during the game. We called this dimension for *goal*. Each dimension were broken further down into three dimension levels, which could be combined to make different treasure hunt games. Furthermore, we needed to create specific elements that were necessary for playing treasure hunt games such as *posts*, *clues*, and *tasks*.

Using an iterative process we were able to create a DSL using the pervasive elements mentioned above. We calculated that it is possible to create a total of 81 different treasure hunt games by combining the dimensions differently. In addition, the game designer has no limitations for how many treasure hunt elements he/she wish to add. The expressiveness could be expanded further by adding more dimensions or levels. However, we feel the current expressiveness is good enough for its intended purpose, which is to make treasure hunt games.

We were able to create other games than just treasure hunt games with our DSL and engine. These games were very similar both in terms of the game mechanics and dynamics. However, it proved that the expressiveness in our DSL is capable to be used outside its purpose.

6.2 Further Work

Due to lack of resources we were not able to complete the DSL and engine for deployment. Neither were we able to play test the treasure hunt games on smart phones in their intended context. This section introduce how we would proceed further and some of the remaining work required on our DSL and engine. It continues by recommending how the DSL could be expanded and used to create other games than treasure hunt games.

6.2.1 DSL and Engine

We estimate that we need at least two more iterations before the DSL and engine can be deployed. First iteration would be to fully implement all the pervasive dimensions with their respective levels. The social dimension still have work remaining. This is due to the fact that it is not possible to create games where the players cooperate with each other. To be more specific, the DSL supports creation of teams, but the engine does not. In addition to this, there are still no mechanics that ranks the players based on their collected points. This is important to implement so that the DSL supports the medium level in the social dimension.

During the second iteration we would try to deploy our code on smart phones, so that the games created with our DSL could be play tested. In order to support these clients we need to create code that runs on smart phones. It would also require a server, so that the clients (smart phones) communicate with the server (engine) as described in section 4.3. In addition, we would have to create a graphical user interface for the clients. We have already created some mockups that shows some important features we find important, see Figures 6.1(a) to 6.1(f).

6.2.2 DSL and Engine Expansion

By increasing the expressiveness in our DSL it would be possible to create more than just treasure hunt games. We recommend that this is done either by dividing one or more of the four pervasive dimensions or to add more levels. In addition to this it might be necessary to create customized elements for the games you wish to create, equivalent to what *posts*, *tasks* and *clues* are for treasure hunt games.

(a) Map view - player move around in real world to find posts.

(b) Map view - Has just found a post, and the associated task pups up.

(c) Map view - player has answered correct, and gets clue for next post.

(d) Status - player can check the game status in any time, by by clicking on the status button.

(e) Posts - player can view all the posts s/he has found, by clicking on the post button.

(f) Clues - player can view all the clues s/he has gathered, by clicking on the clue button.

Figure 6.1: Mockups, illustrating further work concerning mobile clients

Bibliography

[1] Roger Caillois and Meyer Barash. *Man, Play and Games*. University of Illinois Press, 2001.

[2] Michael Dubkov. The future of agile software development: http://www.targetprocess.com/rightthing.html, April 2012.

[3] Eclipse. Eclipse ide: http://www.eclipse.org, April 2012.

[4] Martin Fowler. *A preliminary study on various implementation approaches of domain-specific language*. 2008.

[5] IEEE Architecture Working Group. Ieee std 1471-2000, recommended practice for architectural description of software-intensive systems. Technical report, 2000.

[6] Hong Guo, Hallvard Trætteberg, Alf Inge Wang, and Meng Zhu. Temps: A conceptual framework for pervasive and social games. *Department of Computer and Information Science and NTNU*, 2010.

[7] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. *Design Science in Information Systems Research*. Management Information Systems Research Center, University of Minnesota, 2004.

[8] Habibollah Hosseinpoor. Playtrd architecture for location aware social games. December 2011.

[9] jScience. jscience: http://jscience.org/. April 2012.

[10] Tomaž Kosar, Pablo E. Martínez López, Pablo A. Barrientos, and Marjan Mernik. *Domain Specific Languages*. Addison-Wesley Professional, 2010.

[11] Nicole Lazzaro. Why we play games: Four keys to more emotion without story. *XEODesign, Inc*, 2004.

[12] Ludocity. Pervasive games, street games and new sports: http://ludocity.org/wiki/main_page, April 2012.

[13] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, pages 316–344, 2005.

[14] Markus Montola, Jaakko Stenros, and Annika Waern. Pervasive games: Theory and design: http://pervasivegames.wordpress.com/, April 2012.

[15] Elina M.I. Ollila, Riku Suomela, and Jussi Holopainen. Using prototypes in early pervasive game development. *Nokia*, 2008.

[16] OpenStreetMap. Openstreetmap : http://www.openstreetmap.org. April 2012.

[17] Pacman@Lyon. Pacman@lyon homepage: http://pacmanalyon.net, May 2012.

[18] Peter A. Piccione. In search of the meaning of senet. *Archaeology*, pages 55–58, 1980.

[19] Jennifer Preece, Yvonne Rogers, and Helen Sharp. *Interaction design: Beyond Human-computer Interaction*. John Wiley and Sons Inc, 2002.

[20] Ram Narayanan Sastry. Reducing software complexity: http://ramadvice.wordpress.com, May 2012.

[21] Christian Skar. Model-based software development for pervasive games. December 2011.

[22] Chris Stead. The 10 best game engines of this generation: http://pc.ign.com/articles/100/1003725p1.html, May 2012.

[23] SwingX. Swinglabs swingx : http://java.net/projects/swingx. April 2012.

[24] IPerG team. Iperg homepage: http://iperg.sics.se/tech_space0.php, May 2012.

[25] Pacmanhattan Team. Pacmanhattan homepage: http://pacmanhattan.com, May 2012.

[26] Domain-Specific Language Tools. Visual studio 2008 sdk, domain-specific language tools: http://msdn.microsoft.com/. April 2012.

[27] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. pages 26–36, 2000.

[28] Alf I. Wang, Audrius Jurgelionis, Hong Guo, and Hallvard Trætteberg. Designing enhanced authoring tools for pervasive games. *Mobile Gaming workshop (moga) 2011 on the 8th International Conference on Advances in Computer Entertainment Technology (ACE 2011)*, 2011.

[29] Jeff Ward. What is a game engine?: http://www.gamecareerguide.com/, May 2012.

[30] Wikipedia. Game development: http://en.wikipedia.org/, May 2012.

[31] Wikipedia. Game: http://en.wikipedia.org/, May 2012.

[32] Wikipedia. Level editor: http://en.wikipedia.org/, May 2012.

[33] Wikipedia. List of game engines: http://en.wikipedia.org/, May 2012.

[34] Xtext. The eclipse foundation (xtext) : http://www.eclipse.org/xtext/. April 2012.

Appendix A

Task

Title

DSL and Engine for Pervasive Treasure Hunt Games

Supervisors

Teaching supervisor:
Hallvard Trætteberg

Task Description

Todays mobile clients, with their powerful processing power and built-in technologies, has opened the door for creating pervasive applications that was not possible to create before.

Software complexity and integration still remains a big challenge, and both can be managed by raising the level of abstraction. DSL is a natural way of doing it.

The aim of this master thesis is to create a DSL and an engine for rapid development of pervasive treasure hunt games on mobile clients within the context of Wireless Trondheim.

Appendix B

Game Concepts

This appendix presents the results of our work on domain exploration, where the domain in question is treasure hunt games. This appendix includes several different instances of treasure hunt games, which is described, by means of a simple framework/template.

Introduction

The aim of this iteration was to brainstorm and collect/create a set of treasure hunt game concepts, in order to deduce a common vocabulary (concepts and abstraction specific to treasure hunt games domain) for the creation of the DSL, which is described in the next appendix.

Template

The template below was made in order to make game concepts. They were used to extract concrete game elements that would be included in the domain-specific language for pervasive treasure hunt games.

It is easy to get too focused on details. Try to make an overall description instead of high level of details. (Example: "1st place receives 100 points, 2nd place receives 50 points, ..., nth place receive 1 point" == "The players receives points depending on their position").

ID: A name and/or number to identify this game concept
Short description: A short explanatory description of the game
Game pattern: What is the behavior of the game
Game mechanics:

- Temporal: Determines the time aspect of the game, e.g. when you play for how long

- Social: Human entities that are required in order to play the game
- Mobile: Where is the game played?
- Device(s): Are there any devices required to play the game
- Game objective(s): What is the purpose of playing the game
- Initial: Explains how the game is started
- End: Explains how the game ends
- How to win: Tell how you can win this game and what determines the winner
- How to lose: What determines the looser in the game
- Other: Are there other rules necessary for the game to work as intended

Game play:

- Agon: Are you competing against others?
- Alea: Are there any random aspects in the game?
- Ilinx: Are there any physical activities in the game?
- Mimicry: Does the players, or game, have to mimic anything?
- Paida - Ludus: Describe the level of structure? (structured - non-structured)

Concept Art/Models: Model describing game behavior and the pervasive axis.

Game concept 1

ID: GC1
Short description: This is a simple traditional treasure hunt game, where each individual gets a clue for finding the correct location for the post. When the post is found the player is given the next clue for finding another post. This is repeated until the final post is found, which will end the game.
Game pattern: Treasure hunt game.
Game mechanics:

- Temporal: All the time, no time limit.
- Social: 1 or more persons, the game is played individually.
- Mobile: University campus.
- Device(s): A smart phone with GPS and wireless network.
- Game objective(s): Solve clues given at each post in order to find the next post.

- Initial: The player receives a clue for the first post.
- End: When the last post is found.
- How to win: Find all the posts
- How to lose: Fail to complete the game.
- Other: None

Game play:

- Agon: No real competition, but indirectly against other players.
- Alea: No.
- Ilinx: Yes, walk between posts, looking for posts.
- Mimicry: No.
- Paida - Ludus: Very structured.

Concept Art/Models:

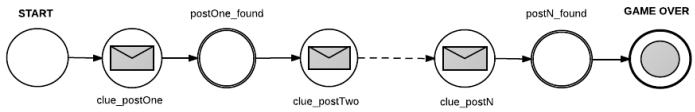


Figure B.1: A model illustrating the behavior of game concept 1

Game concept 2

ID: GC2
Short description: This is a team based treasure hunt game where the players divide themselves in groups of 3. In order to find a post, each group member is given clues for them to find their own sub-post located on different locations. When all member have found their post, they are given partial clues which they have to combine in order to find the post. This is repeated until the final post is found.
Game pattern: Treasure hunt game.
Game mechanics:

- Temporal: All the time, all group members are required to proceed.
- Social: Groups consisting of three players.

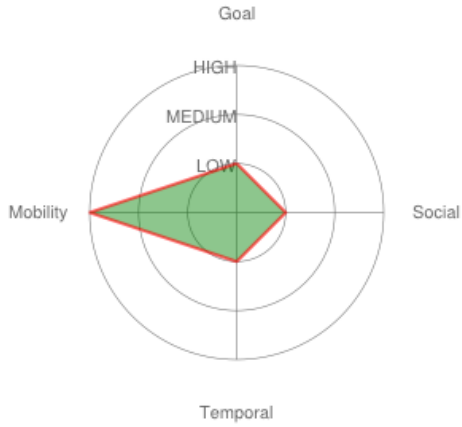


Figure B.2: Game triangle of game concept 1

- Mobile: University campus, Trondheim city (Wireless Trondheim).
- Device(s): A smart phone with GPS and wireless network.
- Game objective(s): Solve clues given at each post in order to find the next post.
- Initial: The groups connect to each other before getting their respective clues.
- End: When the last post is "found" by each group member the game ends.
- How to win: Reaching the last post.
- How to lose: Fail to complete the game.
- Other: None

Game play:

- Agon: Indirectly against other groups.
- Alea: The people in your group (students from your own faculty).
- Ilinx: Yes, walk between posts, looking for posts.
- Mimicry: No.
- Paida - Ludus: Very structured, players have to follow a sequence.

Concept Art/Models:

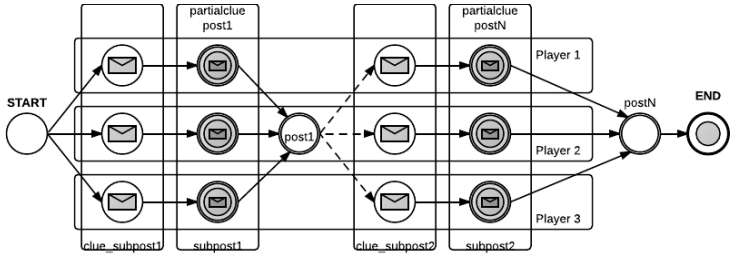


Figure B.3: A model illustrating the behavior of game concept 2

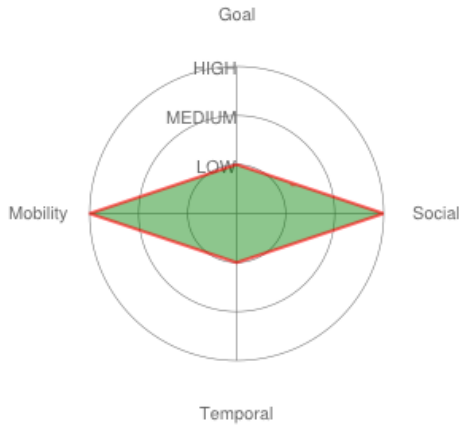


Figure B.4: Game triangle of game concept 2

Game concept 3

ID: GC3
Short description: This is a slightly complicated treasure hunt game where the players are the posts. Each player is given a unique ID or codename which they will give to any player that ask for it. By random, the player will get the location of a post (a player) which he have to find. The game ends when a player has found N posts.
Game pattern: Treasure hunt and racing games.
Game mechanics:

- Temporal: During a defined period.
- Social: Individual, requires a minimum number of players to work.
- Mobile: University campus.

- Device(s): A smart phone with GPS and wireless network.
- Game objective(s): To find all the posts.
- Initial: The players are spread throughout the campus and starts when all players are at their respective spots.
- End: When a player has found N-1 posts, where N is the total number of players.
- How to win: Finding the player number N-1.
- How to lose: If someone else finds N-1 players first.
- Other: None

Game play:

- Agon: Competition between the players.
- Alea: The post order is random.
- Ilinx: Yes, walk between posts, looking for posts.
- Mimicry: No.
- Paida - Ludus: Structured, players have to follow a dynamic sequence.

Concept Art/Models:

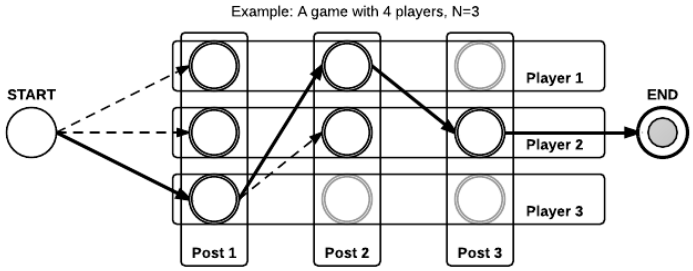


Figure B.5: A model illustrating the behavior of game concept 3

Game concept 4

ID: GC4
Short description: This is a time based treasure hunt game where the players have a time limit to find the posts. If the time runs out, the player has lost and does no longer compete against the others.

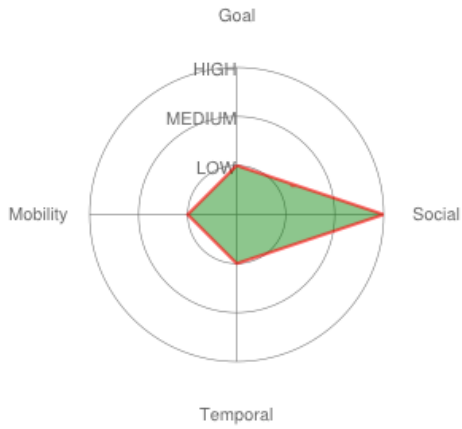


Figure B.6: Game triangle of game concept 3

Game pattern: Treasure hunt and racing games.
Game mechanics:

- Temporal: During a defined period.
- Social: Individual, requires a minimum of players in order to create some competition.
- Mobile: University campus.
- Device(s): A smart phone with GPS and wireless network.
- Game objective(s): To find all the posts within the time limit.
- Initial: The players starts from the same position at the same time.
- End: When the player has found the last post.
- How to win: Finding N posts.
- How to lose: If someone else find N posts first, or the time limit has exceeded.
- Other: None

Game play:

- Agon: Competition between the players.
- Alea: None.
- Ilinx: Yes, walk between posts, looking for posts.

- Mimicry: No.
- Paida - Ludus: Very structured, players have to follow a sequence.

Concept Art/Models:

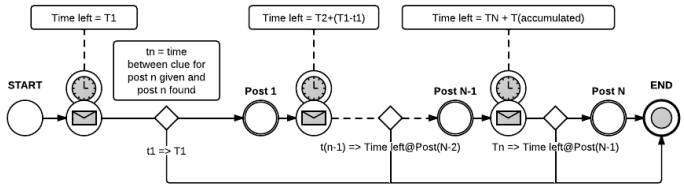


Figure B.7: A model illustrating the behavior of game concept 4

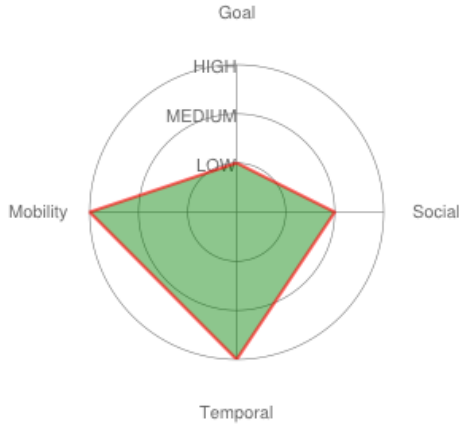


Figure B.8: Game triangle of game concept 4

Game concept 5

ID: TH1
Short description: TH1 is a single player treasure hunt game, where the player is given a map with an approximately positions of the posts marked in the map. In order to complete the game, the player has to find all the posts. Players will be ranked with respect to time, the less time it takes to complete the game the higher is the rank of the player.
Game pattern: Treasure hunt and racing games.
Game mechanics:

- Temporal: When the player starts the game.
- Social: Individual, requires a minimum of players in order to create some competition.
- Mobile: University campus.
- Device(s): A smart phone with GPS and wireless network.
- Game objective(s): To find all the posts.
- Initial: The player starts the game.
- End: When the player has found the last post.
- How to win: Finding all the posts using less time than other players that have completed the game.
- How to lose: If someone else finds all the posts faster.
- Other: None

Game play:

- Agon: Competition between the players.
- Alea: None.
- Ilinx: Yes, walk between posts, looking for posts.
- Mimicry: No.
- Paida - Ludus: Less structured, no determined sequence for finding the posts.

Concept Art/Models:

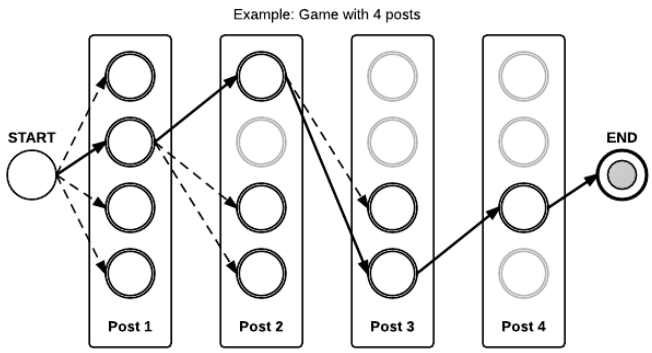


Figure B.9: A model illustrating the behavior of game concept 5

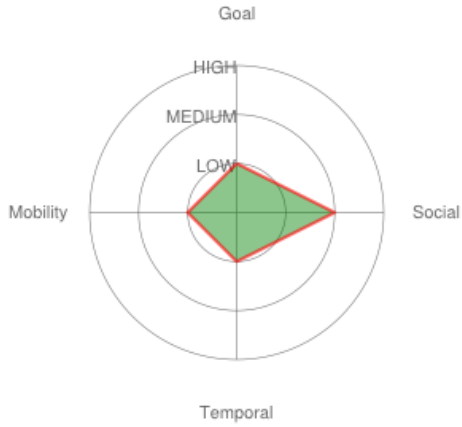


Figure B.10: Game triangle of game concept 5

Game concept 6

ID: TH2
Short description: TH2 is also a single player treasure hunt game, where the player is given a map with an approximately positions of the posts marked in the map. In order to complete the game, the player has to find all the posts, and also solve a task associated with each posts. The task will be given to the player first when s/he finds the associated post. Players will be ranked with respect to number of solved task, then time spent to complete the game. Each task has the same number of points.
Game pattern: Treasure hunt and racing games.
Game mechanics:

- Temporal: When the player starts the game.
- Social: Individual, requires a minimum of players in order to create some competition.
- Mobile: University campus.
- Device(s): A smart phone with GPS and wireless network.
- Game objective(s): To find all the posts and complete their associated tasks as fast as possible.
- Initial: The player starts the game.
- End: When the player has found all the posts and completed their tasks.

- How to win: Finding all the posts and completing their tasks using less time than other players that have completed the game.
- How to lose: If someone else finds all the posts and complete their tasks faster.
- Other: None

Game play:

- Agon: Competition between the players.
- Alea: None.
- Ilinx: Yes, walk between posts, looking for posts.
- Mimicry: No.
- Paida - Ludus: Structured.

Concept Art/Models:

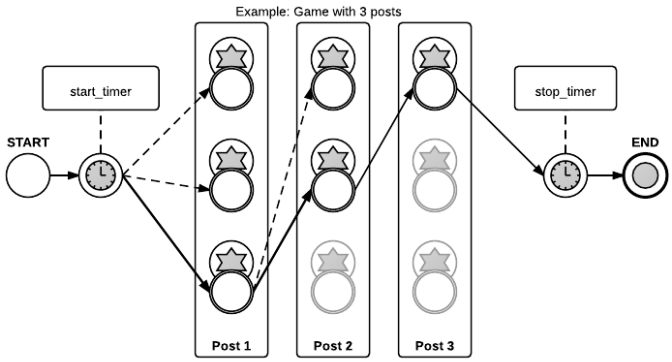


Figure B.11: A model illustrating the behavior of game concept 6

Game concept 7

ID: TH3
Short description: TH3 is also a single player treasure hunt game, where the player is NOT given a map with post positions in advance, but the player is given an initial clue to find the first post. The clue for the next post is found on the first post, and so on. To complete the game the player has to find all the posts. Players will be ranked with respect to time, the less time it takes to complete the game the higher is the rank of the player.
Game pattern: Treasure hunt and racing games.
Game mechanics:

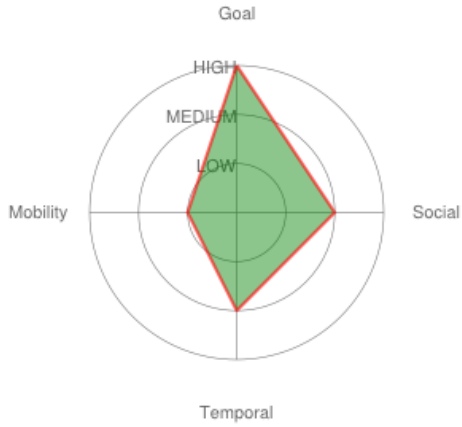


Figure B.12: Game triangle of game concept 6

- Temporal: When the player starts the game.
- Social: 1 or more players, the game is played individually.
- Mobile: University campus.
- Device(s): A smart phone with GPS and wireless network.
- Game objective(s): Solve clues given at each post in order to find the next post.
- Initial: The player starts the game.
- End: When the player has found all the posts.
- How to win: Find all the posts in less time than the other players.
- How to lose: If someone else has completed the game faster.
- Other: None

Game play:

- Agon: Competition between the players.
- Alea: None.
- Ilinx: Yes, walk between posts, looking for posts.
- Mimicry: No.
- Paida - Ludus: Structured.

Concept Art/Models:

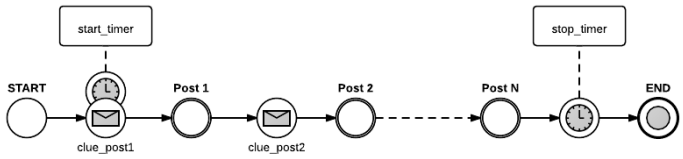


Figure B.13: A model illustrating the behavior of game concept 7

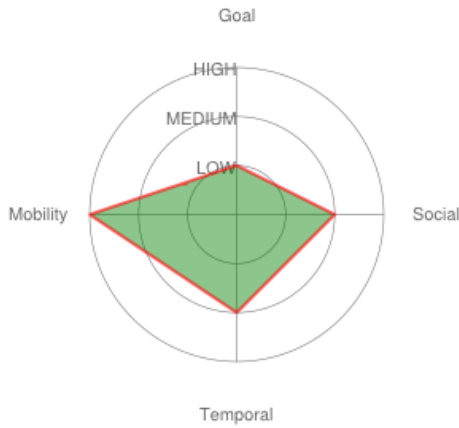


Figure B.14: Game triangle of game concept 7

Game concept 8

ID: TH4
Short description: TH4 is also a single player treasure hunt game, where the player is NOT given a map with post positions in advance, but the player is given an initial clue to find the first post. To complete the game the player has to find all the posts, and also has to solve a task associated with each post. The task will be given to the player first when s/he finds the associated post. And the clue for the next post, is given to the player if and only if the player solves the given task. Players will be ranked with respect to time, the less time it takes to complete the game the higher is the rank of the player.
Game pattern: Treasure hunt and racing games.
Game mechanics:

- Temporal: When the player starts the game.
- Social: 1 or more players, the game is played individually.

- Mobile: University campus.
- Device(s): A smart phone with GPS and wireless network.
- Game objective(s): Solve clues given at each post in order to find the next post and finish their task.
- Initial: The player starts the game.
- End: When the player has found all the posts.
- How to win: Find all the posts in less time than the other players.
- How to lose: If someone else has completed the game faster.
- Other: None

Game play:

- Agon: Competition between the players.
- Alea: None.
- Ilinx: Yes, walk between posts, looking for posts.
- Mimicry: No.
- Paida - Ludus: Structured.

Concept Art/Models:

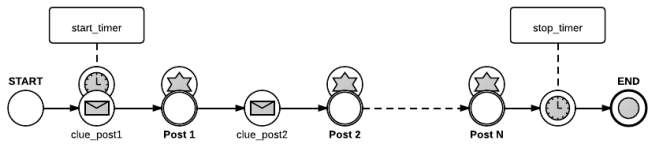


Figure B.15: A model illustrating the behavior of game concept 8

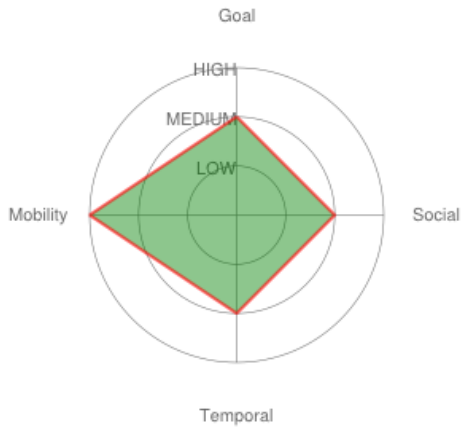


Figure B.16: Game triangle of game concept 8

Appendix C

First Prototype

This appendix presents the results of our first iteration, namely our first prototype of treasure hunt application, which includes the first version of our DSL (.th), first version of our engine (framework) and its graphical user interface (GUI).

Introduction

The aim of this prototype was mainly to discover the capabilities and possibilities of Xtext. Therefore we only implemented the most basic and necessary elements (domain concepts and abstraction), such as player, post, task and clue. As already mentioned in chapter 2.4.1, there are three steps required in order to complete this iteration and produce our first prototype. But we extended this with an extra step (Define/Edit GUI) as the last step. However in this iteration we started with this step, by crated the markups for the engine UI, in order to identify the main functionalities of it.

STEP 0 - Defining GUI

Figure C.1 shows our initial mockup for the GUI. It shows the main functionalities of our treasure hunt application.

C.0.3 Menu panel

Contains main functionalities such as import, export, etc. For instance, from menu File →import, the user can import a new game instance written in our DLS **.th**.

C.0.4 Map area

This area/panel shows the game world. This world consists of (x,y) positions indicated with _, Posts position is indicated with **X**. Players current position is indicated with a character, like **H** for player **Habibollah Hosseinpoor**. Player

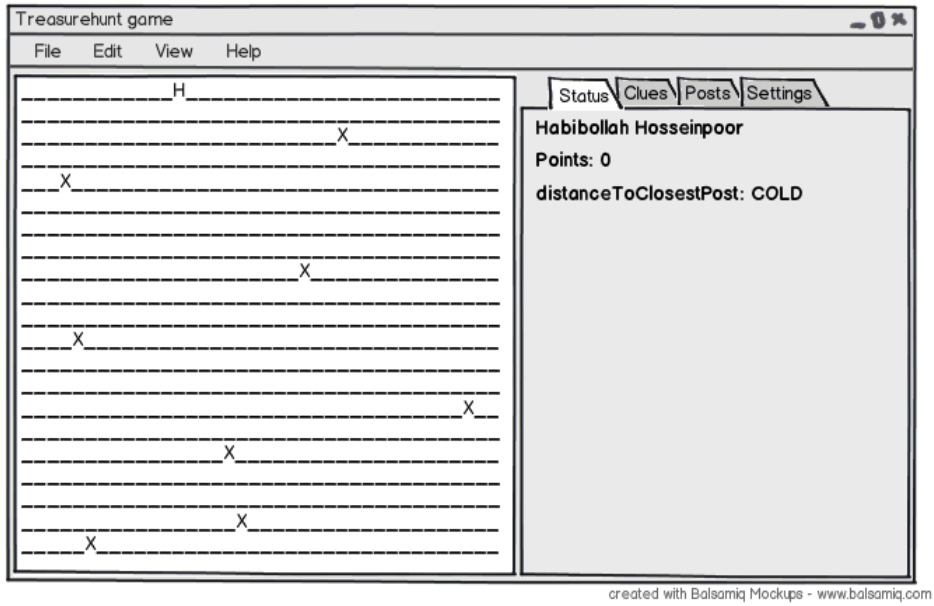


Figure C.1: Treasure hunt UI - first iteration

can move around the world, this can be accomplished on this prototype by clicking on special keys on the keyboard.

C.0.5 Info panels

Shows dynamic game information to the player. For instance, if the player moves close to a post distanceToClosestPost would change depending on the distance to a post from **COLD** to **LUKEWARM, WARM, HOT and FOUND**. This panels contains tabs (**Clues, Posts and Settings**), which hides more functionalities.

Step 1 - Defining the grammar

C.0.6 Treasure hunt DSL .th

The following are our grammar definition (vocabularies) for our first prototype of treasure hunt DSL.

Listing C.1: Grammar Definition for first prototype

```
1 grammar com.habii.treasurehunt.Treasurehunt with org.eclipse.xtext.common.Terminals
2 generate treasurehunt "http://www.habii.com/treasurehunt/Treasurehunt"
3 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
4 TreasureHunt:
5         'Title: ' name=EString
```

```
6         ('goal: ' goal=Goal)
7         ('mapSize: ' width=EInt ',' height=EInt)
8         ('marks: ' defaultMark=Mark ',' visitedMark=Mark ',' postMark=Mark)
9         elements+=AbstractElements*
10 ;
11 AbstractElements:
12     Post | Player;
13 Post:
14     'Post' name=EString '{'
15         'position: ' position=Position
16         ('tasks: ' tasks+=Task (',' tasks+=Task)*)?
17         ('clues: ' clues+=Clue (',' clues+=Clue)*)?
18         ('visitedBy: ' visitedBy+=[Player|EString] (',' visitedBy+=[Player|EString])
               *)?
19     '}'
20 ;
21 Player:
22     'Player' name=EString '{'
23         'initialClue: ' initialClue=[Clue|QID]
24         'distanceToClosestPost: ' distanceToClosestPost=DistanceToClosestPost
25         ('visited: ' visited+=[Post|EString] (',' visited+=[Post|EString])*)?
26         ('solved: ' solved+=[Task|QID] (',' solved+=[Task|QID])*)?
27         ('currentPosition: ' currentPosition=Position)?
28         ('direction: ' direction=Direction)?
29         ('mark: ' mark=Mark)?
30     '}'
31 ;
32 QID:
33     (ID|STRING) ('.'(ID|STRING))*;
34 Position:
35     GPS | WLAN;
36 GPS:
37     'GPS: ' '(' x=EInt ', ' y=EInt ')';
38 WLAN:
39     'WLAN: ' '(' x=EInt ', ' y=EInt ')';
40 Task:
41     (name=ID)? question=EString'?' answer=EString'!' ('point: ' point=EInt)?;
42 Clue:
43     (name=ID)? clueText=EString;
44 enum Goal:
45     SOLVE = 'SOLVE' | FIND = 'FIND' | FINDANDSOLVE='FINDANDSOLVE';
46 enum Direction:
47     NORTH = 'W' | SOUTH = 'S' | EAST = 'D' | WEST = 'A';
48 enum DistanceToClosestPost:
49     FOUND = 'FOUND' | HOT = 'HOT' | WARM = 'WARM' | LUKEWARM = 'LUKEWARM' | COLD = '
           COLD';
50 EString returns ecore::EString:
51     STRING | ID;
52 EInt returns ecore::EInt:
53     '-'? INT;
54 terminal Mark:
55     ('a'..'z'|'A'..'Z'|'_'|'0'..'9');
```

C.0.7 Example Game instance for Treasure hunt DSL .th

The following shows an instance of our first prototype Treasure hunt DSL (.th).

Listing C.2: Treasure hunt DSL sample script

```
Title: "Advanture"
mapSize: 30,50
goal: FIND

Post "Native Village" {
       position: GPS: (02, 49)
       clues: c1 "Find the most beutiful lake in the world..."
}
Post "Paradice Lake" {
       position: GPS: ( 03, 21)
       clues: c2 "Find the snow..."
}
Post "Cold Mountain" {
       position: WLAN: ( 13, 14)
       clues: c3 "Find the the forest no man has crossed alive..."
}
Post "Fearsome Forest" {
       position: GPS: ( 14, 45)
       clues: c4 "Clue to Post4"
}
Post "Never Ending River" {
       position: GPS: ( 29, 16)
       clues: c5 "Go there where do turist go on vacation..."
}

Post "Pine Trees Beach" {
       position: WLAN: ( 21, 34)
       clues: c6 "Where do pirates hang out?..."
}
Post "Pirate Cave" {
       position: GPS: ( 10, 09)
       clues: c7 "Find the most famous water falls..."
}
Post "Niagra Falls" {
       position: GPS: ( 27, 49)
       clues: c8 "Where do native people live..."
}

// Players
Player Skar {
       initialClue: "Never Ending River".c5
       distanceToClosestPost: COLD
       currentPosition: GPS: ( 05, 05)
       direction: D
       mark: H
}
```

Step 2 - Run the generator

As mentioned briefly in chapter 2.4.1, Xtext will drive various language components in this step. It generates the parser and serializer and some additional infrastructure code.

Figure C.2 shows a class diagram of the generated models, which are implemented using the Eclipse Modeling Framework (EMF), as can be seen as a very powerful version of JavaBeans.

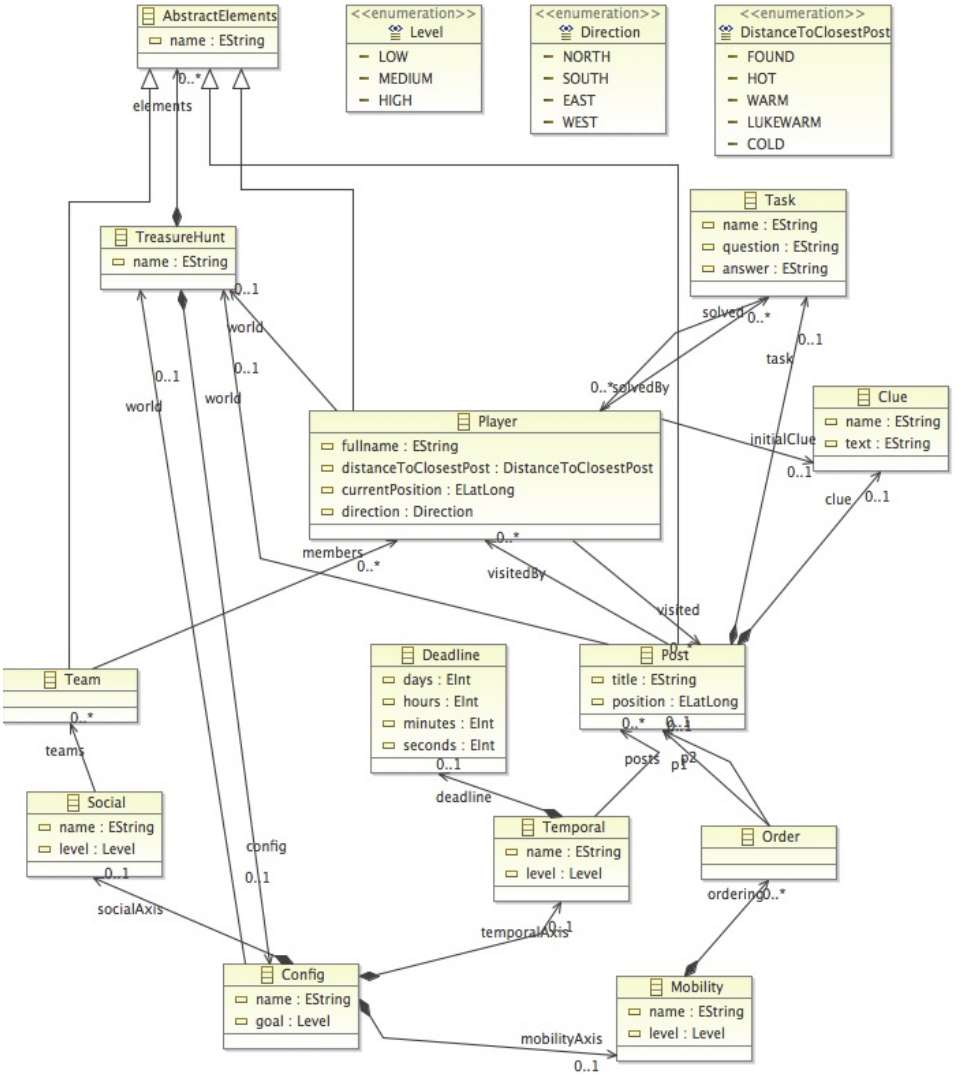


Figure C.2: EMF models for third prototype

Also, in this step an editor see Figure C.3 for writing models / instances in the newly created DLS is generated. The editor is pretty sophisticated, it comes equipped with code completion, syntax coloring, custom keyword coloring, real time constraints, validation and more.

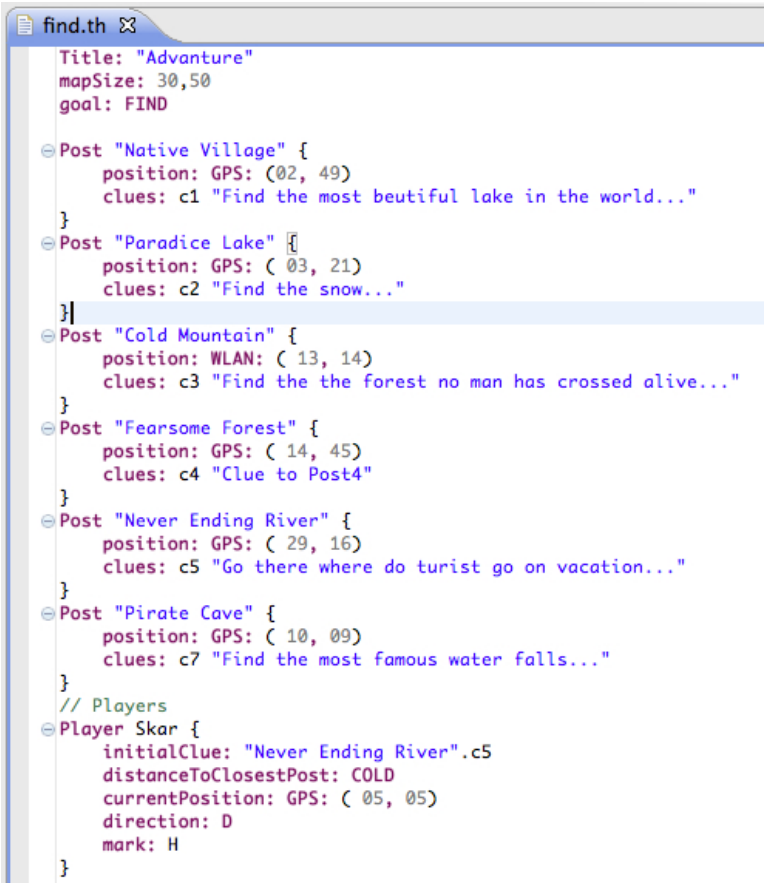


Figure C.3: Xtext generated editor for treasure hunt DSL (.th)

Step 3 - Defining/writing the engine

A DSL isn't worth much if you are not able to execute it somehow. Since text files parsed by Xtext are represented as object graphs in memory, also called Abstract Syntax Tree (AST), we can load them at runtime in order to use them dynamically.

C.0.8 Loading a Resource

EMF models can be persisted by the means of a so called Resource. Xtext languages implement the Resource interface, thus allowing us to use the EMF API to load a model into memory and do whatever we want with it:

Listing C.3: Using EMF API

```
1 new DomainmodelStandaloneSetup().createInjectorAndDoEMFRegistration();
2
```

```
ResourceSet rs = new ResourceSetImpl();
Resource resource = rs.getResource(URI.createURI("./adventure.th"), true);
EObject eobject = resource.getContents().get(0);
```

C.0.9 Working with EMF Models

In order to increase the level of abstraction and to simplify the our DSL as much as possible, we put all the unnecessary code into the engine. We also extended the functionalities of EMF generated objects, by created subclasses and change the EMF Factory class to return instances of the extended classes see Figure C.4 and Figure C.5 instead:

Listing C.4: Working with EMF models

```
TreasurehuntPackageFactory.createTreasurehuntImp(){
     World treasureHunt = new World();
    return treasureHunt;
}

TreasurehuntPackageFactory.createPlayerImp(){
        EnginePlayer player = new EnginePlayer();
    return player;
}
```

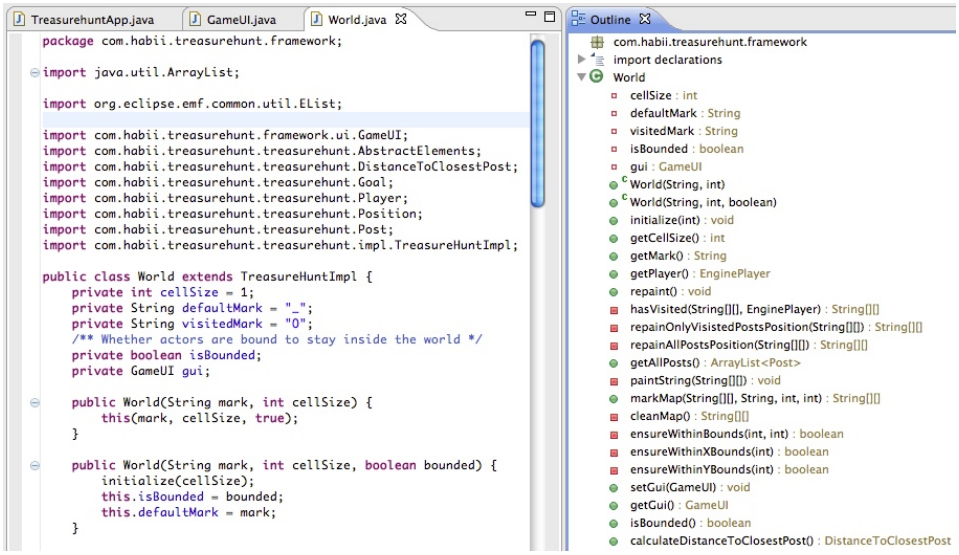
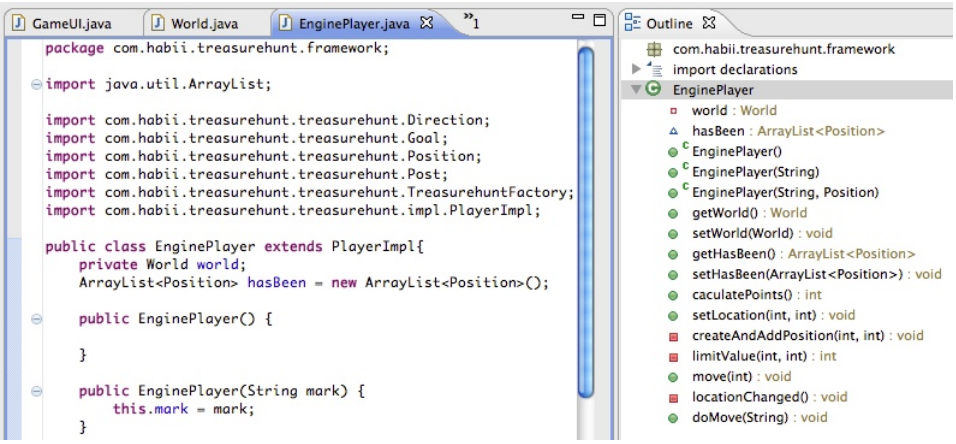


Figure C.4: World

How It all works out

Figure C.6, shows how it all works once everything is implemented.




```
package com.habii.treasurehunt.framework;

import java.util.ArrayList;

import com.habii.treasurehunt.treasurehunt.Direction;
import com.habii.treasurehunt.treasurehunt.Goal;
import com.habii.treasurehunt.treasurehunt.Position;
import com.habii.treasurehunt.treasurehunt.Post;
import com.habii.treasurehunt.treasurehunt.TreasurehuntFactory;
import com.habii.treasurehunt.treasurehunt.impl.PlayerImpl;

public class EnginePlayer extends PlayerImpl{
    private World world;
    ArrayList<Position> hasBeen = new ArrayList<Position>();

    public EnginePlayer() {

    }

    public EnginePlayer(String mark) {
        this.mark = mark;
    }
```

Figure C.5: EnginePlayer

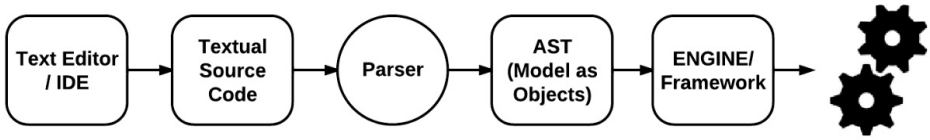


Figure C.6: Overview of steps crating a model in .th

Text Editor / IDE Is used to create models / game instances of treasure hunt DSL (.th),

Textual Source Code Once the model / game instance is written, it is represented in Figure C.6 as textual source code, which is given to the parser as input.

Parser Parses the input (Textual source code) and produces an Abstract Syntax Tree (AST), in our case Xtext produces also a complete model as java Object by using EMF.

AST (Model as Objects) The output of the Parser is a set of generated java objects.

Engine / Framework We extend the generated java objects from the previous step in order to function as the defined semantic, and we call it Engine.

Runtime Once it all is lunched it works as intended.

Main functionalities covered in this prototype

C.0.10 .th

Our DSL is called Treasure hunt and it has got the file extension .th, see subsection C.0.7 for more detail overview of the .th DSL.

C.0.11 Grammar definition

In this prototype we have created a simple grammar definition by using Xtext, to support creation of a game instance (model). See subsection C.0.6 for more detail overview of the grammar definition.

C.0.12 Engine

In this prototype we managed to build a simple **Engine** in order to process the model written in **.th**. Since Xtext models are implemented using the Eclipse Modeling Framework (EMF), this reduces the complexity of our Engine greatly.

Our Engine basically consists of two java classes, and lots of UI stuff.

C.0.13 Importing game instances

In this prototype we have also implemented import/export of game instances / models written in **.th**. We are able to save a game by exporting it to a file.

Figure C.7, shows our engine prototype during the execution of an instances of treasure hunt game.

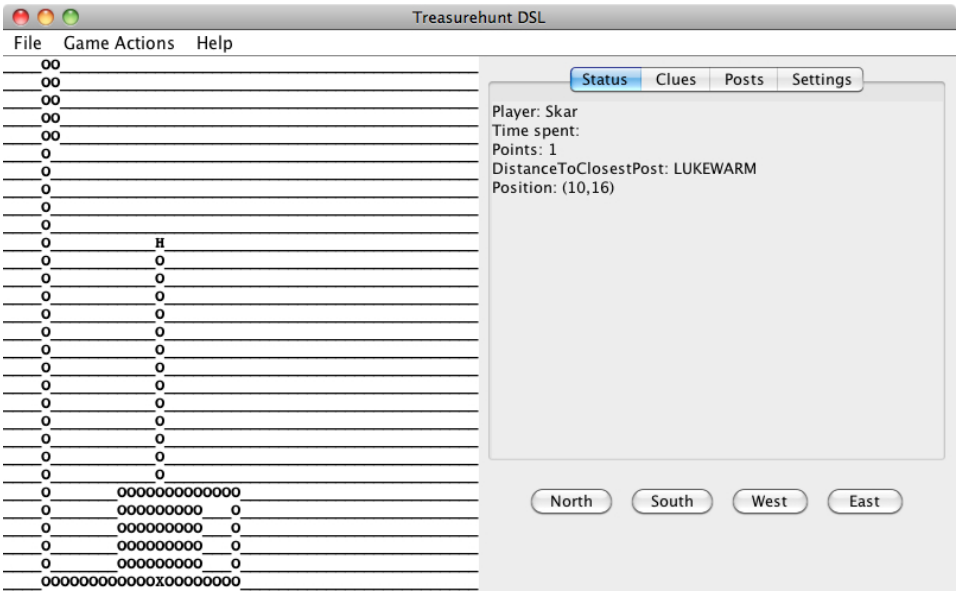


Figure C.7: Treasure hunt first prototype

Appendix D

Second Prototype

This appendix presents the results of our second iteration, which is basically implementing a real world map, with real coordinates in our prototype.

Introduction

The aim of this prototype was mainly to implement a real world map by using software components such as jScience, swingX and OpenStreetMap.

Step 1 - Redefining the grammar

D.0.14 Treasure hunt DSL(v2) .th

The following are our grammar definition for our second prototype. Only one new concepts are introduced in this iteration (Strategy), which is actually the result of re-factorization. But now we can add read coordinates (latitude and longitude).

Listing D.1: Grammar Definition for second prototype

```
1  grammar com.habii.treasurehunt.Treasurehunt with org.jscience.xtext.Jscience
2  generate treasurehunt "http://www.habii.com/treasurehunt/Treasurehunt"
3  import "http://www.eclipse.org/emf/2002/Ecore" as ecore
4
5  TreasureHunt:
6      'World: ' name=EString
7      strategy=Strategy
8      elements+=AbstractElements*
9  ;
10 Strategy:
11     'Strategy' name=EString '{'
12        'goal: ' goal=Level
13     '}'
14 ;
15 AbstractElements:
16     Post | Player
```

```
17 ;
18 Post:
19     'Post' name=EString '{'
20         'position: ' position=ELatLong
21         ('tasks: ' tasks+=Task (',' tasks+=Task)*)?
22         ('clues: ' clues+=Clue (',' clues+=Clue)*)?
23         ('visitedBy: ' visitedBy+=[Player|EString] (',' visitedBy+=[Player|EString])
               *)?
24     '}'
25 ;
26 Player:
27     'Player' name=EString '{'
28         'initialClue: ' initialClue=[Clue|QID]
29         'distanceToClosestPost: ' distanceToClosestPost=DistanceToClosestPost
30         ('visited: ' visited+=[Post|EString] (',' visited+=[Post|EString])*)?
31         ('solved: ' solved+=[Task|QID] (',' solved+=[Task|QID])*)?
32         ('currentPosition: ' currentPosition=ELatLong)?
33         ('direction: ' direction=Direction)?
34         ('world: ' world=[TreasureHunt])?
35     '}'
36 ;
37 QID:
38     EString ('.' EString)*
39 ;
40 Task:
41     name=EString'?' answer=EString'!'
42     ('solvedBy: ' solvedBy+=[Player|EString] (',' solvedBy+=[Player|EString])*)?
43 ;
44 Clue:
45     name=EString
46 ;
47 enum Level:
48     LOW = 'LOW' | MEDIUM = 'MEDIUM' | HIGH = 'HIGH'
49 ;
50 enum Direction:
51     NORTH = 'W' | SOUTH = 'S' | EAST = 'D' | WEST = 'A'
52 ;
53 enum DistanceToClosestPost:
54     FOUND = 'FOUND' | HOT = 'HOT' | WARM = 'WARM' | LUKEWARM = 'LUKEWARM' | COLD = '
           COLD'
55 ;
56 EString returns ecore::EString:
57     STRING | ID;
58 EInt returns ecore::EInt:
59     '-'? INT;
60 terminal Mark:
61     ('a'..'z'|'A'..'Z'|'_'|'0'..'9');
```

Step 2 - Run the generator

This step is the same in all the iteration/prototype, since it just generates artifacts according to the grammar definition (step 1).

D.0.15 Example Game instance for Treasure hunt DSL(v2) .th

The following shows an instance of our second prototype Treasure hunt DSL (.th).

Listing D.2: Treasure hunt DSL (v2) sample script

```
World: Advanture
Strategy Simple {
       goal: MEDIUM
}
// Posts
Post "Native Village" {
       position: 63.427712,10.37607
       clues: "Find the most beautiful lake in the world..."
}
Post "Fearsome Forest" {
       position: 63.420723,10.39993
       tasks: "How many trees are in this forest"? "1000000"!
       clues: "Clue to Post4"
}
Post "Pirate Cave" {
       position: 63.43324,10.354614
       tasks: "Hwo is the pirate king"? "Jack"!
       clues: "Find the most famous water falls..."
}
// Players
Player Habibollah {
       initialClue: "Never Ending River"."Go there where do tourist go on vacation
            ..."
       distanceToClosestPost: COLD
       currentPosition: 63.430476,10.39255
       direction: D
       world: Advanture
}
Player Christian {
       initialClue: "Niagara Falls"."Where do native people live..."
       distanceToClosestPost: LUKEWARM
       currentPosition: 63.42669999999989,10.389000000000003
       direction: A
       world: Advanture
}
```

Step 3 - Updating the engine

The biggest change to engine in this iteration was to the implementation of jScience, swingX and OpenStreetMap.

D.0.16 jScience

Used for real Latitude and Longitude values in order to integrated a real world map in a map viewer. We simple used the LatLong class provided by the jScience,

which besides taking Latitude and longitude value also takes in a Unit in this case Degree as opposed to Radian.

Listing D.3: Integrating jScience

```
1 LatLong latLong = LatLong.valueOf(34.34, 23.56, DEGREE_ANGLE);
```

D.0.17 SwingX

Main functionalities:

- paint all players on the map.
- paint all posts on the map.
- show post info, it it is clicked on.
- show player info, if the player is clicked on the map
- move player on the map by keys (a,w,s,d).

Listing D.4: Integrating SwingX, OpenStreetMap and jScience

```
1  package com.habii.treasurehunt.framework.ui;
2
3  import java.awt.Point;
4  import java.awt.Rectangle;
5  import java.awt.event.KeyEvent;
6  import java.awt.event.KeyListener;
7  import java.awt.event.MouseEvent;
8  import java.awt.event.MouseListener;
9  import java.awt.geom.Point2D;
10 import java.util.HashMap;
11 import java.util.Set;
12
13 import javax.measure.unit.NonSI;
14 import javax.swing.JOptionPane;
15 import javax.swing.ToolTipManager;
16 import javax.swing.event.EventListenerList;
17
18 import org.jdesktop.swingx.JXMapKit;
19 import org.jdesktop.swingx.JXMapViewer;
20 import org.jdesktop.swingx.mapviewer.GeoPosition;
21 import org.jdesktop.swingx.mapviewer.Waypoint;
22 import org.jdesktop.swingx.mapviewer.WaypointPainter;
23 import org.jscience.geography.coordinates.LatLong;
24
25 import com.habii.treasurehunt.framework.EnginePlayer;
26 import com.habii.treasurehunt.framework.EngineWorld;
27 import com.habii.treasurehunt.framework.ui.editor.AbstractEditor;
28 import com.habii.treasurehunt.framework.ui.editor.PlayerEditor;
29 import com.habii.treasurehunt.framework.ui.editor.PostEditor;
30 import com.habii.treasurehunt.framework.ui.map.BaseWaypoint;
31 import com.habii.treasurehunt.framework.ui.map.BaseWaypointPainter;
32 import com.habii.treasurehunt.framework.ui.map.PlayerWayPoint;
```

```
33 import com.habii.treasurehunt.framework.ui.map.PostWayPoint;
34 import com.habii.treasurehunt.treasurehunt.Post;
35 import com.habii.treasurehunt.treasurehunt.Task;
36
37 public class MapPanel extends JXMapKit{
38         private static final long serialVersionUID = 1787118143488776032L;
39         public final static double DEFUALT_LATITUDE = 63.430476;
40         public final static double DEFAULT_LONGITUDE = 10.39255;
41         public final int DEFAULT_ZOOM = 3;
42
43         private CardPanel editors;
44         private HashMap<String, AbstractEditor> cardMap;
45
46         private EnginePlayer selectedPlayer;
47         private Post selectedPost;
48
49         protected EventListenerList _waypointEventListeners = new EventListenerList()
               ;
50
51         protected Set<BaseWaypoint> waypoints;
52         private JXMapViewer map;
53
54         public MapPanel(CardPanel editors, HashMap<String, AbstractEditor> cardMap){
55                 init(DEFUALT_LATITUDE, DEFAULT_LONGITUDE);
56                 this.editors = editors;
57                 this.cardMap = cardMap;
58         }
59
60         private void init(double latitude, double longitude){
61                 setDefaultProvider(DefaultProviders.OpenStreetMaps);
62                 setCenterPosition(new GeoPosition(latitude, longitude));
63                 setZoom(DEFAULT_ZOOM);
64                 setMiniMapVisible(false);
65                 setZoomButtonsVisible(true);
66                 setZoomSliderVisible(false);
67
68                 ToolTipManager.sharedInstance().setInitialDelay(0);
69
70                 map = getMainMap();
71
72         map.setZoomEnabled(false);
73         map.setPanEnabled(true);
74                 map.addMouseListener(new MouseListener(){
75                         @Override
76                         public void mouseClicked(MouseEvent event) {
77                                 Waypoint waypoint = getWaypoint(event.getPoint());
78                 if (waypoint != null){
79                         if(waypoint instanceof PlayerWayPoint){
80                                 playerWayPointUpdate(waypoint);
81                         } else if (waypoint instanceof PostWayPoint){
82                                 postWayPointUpdate(waypoint);
83                         }
84                 } else {
85                         PostEditor editor = (PostEditor) cardMap.get(CardPanel.POST);
86                         Post post = editor.getSelectedPost();
87
88                         if(post!=null){
```

```
89                              LatLong coordinates = getCoordinates(event.getPoint());
90                              post.setPosition(coordinates);
91                              EngineWorld world = (EngineWorld) post.getWorld();
92                                              world.updateMap();
93                              editor.update(post);
94                      }
95              }
96                      }
97
98                      @Override
99                      public void mouseEntered(MouseEvent event) {}
100                     @Override
101                     public void mouseExited(MouseEvent event) {}
102                     @Override
103                     public void mousePressed(MouseEvent event) {}
104                     @Override
105                     public void mouseReleased(MouseEvent event) {}
106             });
107
108             map.addKeyListener(new KeyListener() {
109                     @Override
110                     public void keyTyped(KeyEvent e) {
111                             char keyChar = e.getKeyChar();
112                             if(keyChar=='w'){
113                                     doMove("W");
114                             } else if(keyChar=='s'){
115                                     doMove("S");
116                             } else if(keyChar=='a'){
117                                     doMove("A");
118                             } else if(keyChar=='d'){
119                                     doMove("D");
120                             } else {
121                             }
122                     }
123                     @Override
124                     public void keyReleased(KeyEvent e) { }
125                     @Override
126                     public void keyPressed(KeyEvent e) { }
127             });
128
129     }
130
131     public LatLong getCoordinates(Point mousePoint){
132             GeoPosition centerPosition = getMainMap().getCenterPosition();
133             double latitude = centerPosition.getLatitude();
134             double longitude = centerPosition.getLongitude();
135             return LatLong.valueOf(latitude, longitude, NonSI.DEGREE_ANGLE);
136     }
137
138     private void playerWayPointUpdate(Waypoint waypoint) {
139             PlayerWayPoint playerWayPoint = (PlayerWayPoint) waypoint;
140             selectedPlayer = (EnginePlayer) playerWayPoint.getPlayer();
141             editors.showPlayerPanel();
142             PlayerEditor playerEditor = (PlayerEditor) cardMap.get(CardPanel.
                    PLAYER);
143             playerEditor.update(selectedPlayer);
144
```

```
145         }
146
147         private void postWayPointUpdate(Waypoint waypoint) {
148                PostWayPoint postWayPoint = (PostWayPoint) waypoint;
149                selectedPost = postWayPoint.getPost();
150
151                editors.showPostPanel();
152                PostEditor postEditor = (PostEditor) cardMap.get(CardPanel.POST);
153                postEditor.update(selectedPost);
154         }
155
156         @SuppressWarnings("all")
157         public void setWaypoints(Set<BaseWaypoint> waypoints){
158                this.waypoints = waypoints;
159                WaypointPainter painter = new BaseWaypointPainter();
160            painter.setWaypoints(waypoints);
161            getMainMap().setOverlayPainter(painter);
162         }
163
164         public Waypoint getWaypoint(Point mousePoint){
165                if (waypoints == null){
166                       return null;
167                }
168
169                JXMapViewer map = getMainMap();
170
171                Rectangle bounds = map.getViewportBounds();
172                for(BaseWaypoint waypoint : waypoints){
173                       Point2D point = map.getTileFactory().geoToPixel(waypoint.
                              getPosition(), map.getZoom());
174
175                       int x = (int)(point.getX() - bounds.getX());
176            int y = (int)(point.getY() - bounds.getY());
177            int s = waypoint.getSize();
178
179            if (new Rectangle(x - s, y - s, s * 2, s * 2).contains(mousePoint)){
180                return waypoint;
181            }
182                }
183
184                return null;
185         }
186
187         public void addWayPoint(BaseWaypoint waypoint){
188                waypoints.add(waypoint);
189         }
190
191         public void cleanWayPoints(){
192                waypoints.clear();
193         }
194
195         public EnginePlayer getSelectedPlayer() {
196                return selectedPlayer;
197         }
198
199         public Post getSelectedPost() {
200                return selectedPost;
```

```
201         }
202
203         private void doMove(String cmd) {
204                 if(selectedPlayer==null){
205                         return;
206                 }
207                 selectedPlayer.doMove(cmd);
208                 Post post = selectedPlayer.calculateDistanceToClosestPost();
209                 if(post!=null){
210                         Task t = post.getTask();
211                         String answer = JOptionPane.showInputDialog(t.getQuestion()+"?
                                "");
212                         if(t.getAnswer().equalsIgnoreCase(answer)){
213                                 if(selectedPlayer.calculateStopWatch()>0){
214                                         selectedPlayer.solved(t);
215                                         t.getSolvedBy().add(selectedPlayer);
216                                 }
217                         }
218                 }
219
220                 EngineWorld world = (EngineWorld) selectedPlayer.getWorld();
221                 world.updateMap();
222                 world.updateEditor(selectedPlayer);
223         }
224 }
```

Open Street Map

is a collaborative project to create a free editable map of the world. We use this as our map provider, for our engine UI.

Listing D.5: Integrating OpenStreetMap

```
1 setDefaultProvider(DefaultProviders.OpenStreetMaps);
```

The app UI

Figure C.7, shows our second engine prototype during the execution of an instances of treasure hunt game. A player object can now be moved around in the map by clicking on the object first, then clicking on one of the four buttons (North, Sought, East and West).

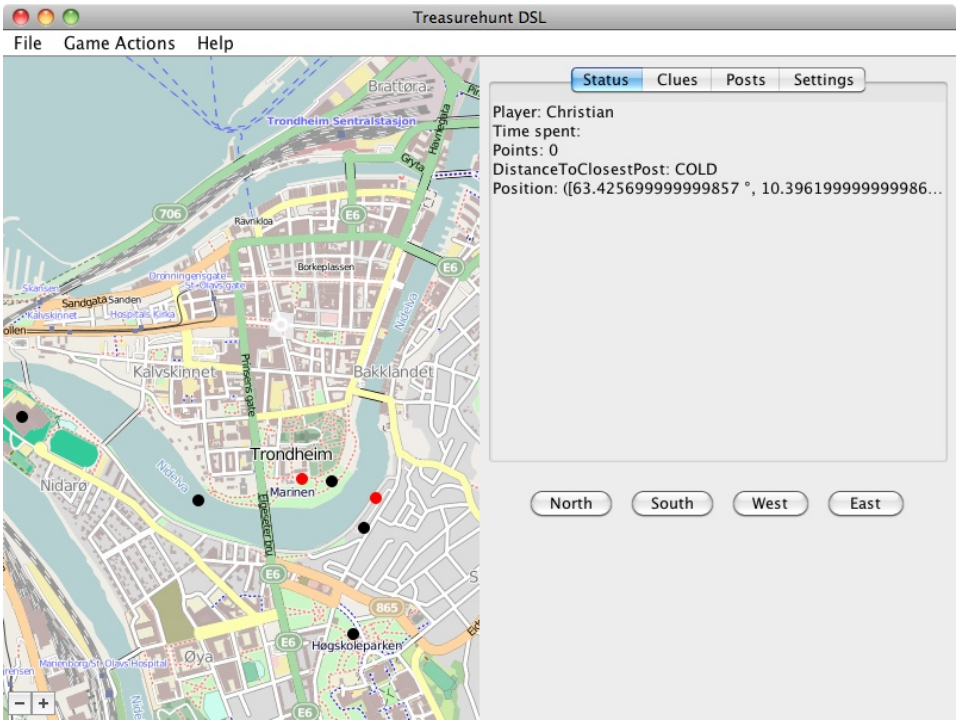


Figure D.1: Treasure hunt second prototype

Appendix E

Third Prototype

This appendix presents the results of our third iteration/prototype of treasure hunt application, which includes the third version of our DSL (.th), third version of our engine (framework) and its GUI.

Introduction

The aim of this prototype was on our DSL expressiveness, this includes defining and implementing completely orthogonal pervasive axis, such as **Mobility**, **Temporal**, and **Social**, thus allowing us to create different games.

DSL expressiveness

As mentioned in Chapter 4, we defined three orthogonal pervasive dimensions **Mobility**, **Temporal**, and **Social**. Where for each of this axis we defined three different levels, **LOW**, **MEDIUM** and **HIGH**, that has a particular meaning for each dimensions.

Table E.1, shows a compact view of all the dimensions and their levels, along with a brief descriptions.

When creating a treasure hunt game the developer can combine this pervasive dimensions. Figure E.1 is once instance of this combination of dimensions, illustrated by a radar diagram.

As Table E.2, shows we are able to create 81 unique game types by combining the dimensions, thus giving the developer of treasure hunt games lots of flexibility.

Game ID	Goal	Mobility	Social	Temporal
GI01	LOW	LOW	LOW	LOW
GI02	LOW	LOW	LOW	MEDIUM
GI03	LOW	LOW	LOW	HIGH
GI04	LOW	LOW	MEDIUM	LOW

GI05	LOW	LOW	MEDIUM	MEDIUM
GI06	LOW	LOW	MEDIUM	HIGH
GI07	LOW	LOW	HIGH	LOW
GI08	LOW	LOW	HIGH	MEDIUM
GI09	LOW	LOW	HIGH	HIGH
GI10	LOW	MEDIUM	LOW	LOW
GI11	LOW	MEDIUM	LOW	MEDIUM
GI12	LOW	MEDIUM	LOW	HIGH
GI13	LOW	MEDIUM	MEDIUM	LOW
GI14	LOW	MEDIUM	MEDIUM	MEDIUM
GI15	LOW	MEDIUM	MEDIUM	HIGH
GI16	LOW	MEDIUM	HIGH	LOW
GI17	LOW	MEDIUM	HIGH	MEDIUM
GI18	LOW	MEDIUM	HIGH	HIGH
GI19	LOW	HIGH	LOW	LOW
GI20	LOW	HIGH	LOW	MEDIUM
GI21	LOW	HIGH	LOW	HIGH
GI22	LOW	HIGH	MEDIUM	LOW
GI23	LOW	HIGH	MEDIUM	MEDIUM
GI24	LOW	HIGH	MEDIUM	HIGH
GI25	LOW	HIGH	HIGH	LOW
GI26	LOW	HIGH	HIGH	MEDIUM
GI27	LOW	HIGH	HIGH	HIGH
GI28	MEDIUM	LOW	LOW	LOW
GI29	MEDIUM	LOW	LOW	MEDIUM
GI30	MEDIUM	LOW	LOW	HIGH
GI31	MEDIUM	LOW	MEDIUM	LOW
GI32	MEDIUM	LOW	MEDIUM	MEDIUM
GI33	MEDIUM	LOW	MEDIUM	HIGH
GI34	MEDIUM	LOW	HIGH	LOW
GI35	MEDIUM	LOW	HIGH	MEDIUM
GI36	MEDIUM	LOW	HIGH	HIGH
GI37	MEDIUM	MEDIUM	LOW	LOW
GI38	MEDIUM	MEDIUM	LOW	MEDIUM
GI39	MEDIUM	MEDIUM	LOW	HIGH
GI40	MEDIUM	MEDIUM	MEDIUM	LOW
GI41	MEDIUM	MEDIUM	MEDIUM	MEDIUM
GI42	MEDIUM	MEDIUM	MEDIUM	HIGH
GI43	MEDIUM	MEDIUM	HIGH	LOW
GI44	MEDIUM	MEDIUM	HIGH	MEDIUM
GI45	MEDIUM	MEDIUM	HIGH	HIGH
GI46	MEDIUM	HIGH	LOW	LOW
GI47	MEDIUM	HIGH	LOW	MEDIUM
GI48	MEDIUM	HIGH	LOW	HIGH
GI49	MEDIUM	HIGH	MEDIUM	LOW

GI50	MEDIUM	HIGH	MEDIUM	MEDIUM
GI51	MEDIUM	HIGH	MEDIUM	HIGH
GI52	MEDIUM	HIGH	HIGH	LOW
GI53	MEDIUM	HIGH	HIGH	MEDIUM
GI54	MEDIUM	HIGH	HIGH	HIGH
GI55	HIGH	LOW	LOW	LOW
GI56	HIGH	LOW	LOW	MEDIUM
GI57	HIGH	LOW	LOW	HIGH
GI58	HIGH	LOW	MEDIUM	LOW
GI59	HIGH	LOW	MEDIUM	MEDIUM
GI60	HIGH	LOW	MEDIUM	HIGH
GI61	HIGH	LOW	HIGH	LOW
GI62	HIGH	LOW	HIGH	MEDIUM
GI63	HIGH	LOW	HIGH	HIGH
GI64	HIGH	MEDIUM	LOW	LOW
GI65	HIGH	MEDIUM	LOW	MEDIUM
GI66	HIGH	MEDIUM	LOW	HIGH
GI67	HIGH	MEDIUM	MEDIUM	LOW
GI68	HIGH	MEDIUM	MEDIUM	MEDIUM
GI69	HIGH	MEDIUM	MEDIUM	HIGH
GI70	HIGH	MEDIUM	HIGH	LOW
GI71	HIGH	MEDIUM	HIGH	MEDIUM
GI72	HIGH	MEDIUM	HIGH	HIGH
GI73	HIGH	HIGH	LOW	LOW
GI74	HIGH	HIGH	LOW	MEDIUM
GI75	HIGH	HIGH	LOW	HIGH
GI76	HIGH	HIGH	MEDIUM	LOW
GI77	HIGH	HIGH	MEDIUM	MEDIUM
GI78	HIGH	HIGH	MEDIUM	HIGH
GI79	HIGH	HIGH	HIGH	LOW
GI80	HIGH	HIGH	HIGH	MEDIUM
GI81	HIGH	HIGH	HIGH	HIGH

Table E.2: Combination of axis

Step 1 - Defining the grammar

E.0.18 Treasure hunt (v3) DSL .th

The following are our grammar definition for our third prototype. The most important concepts defined in this iteration are the pervasive dimensions as already described.

	LOW	MEDIUM	HIGH	Constraints
Social	single player	Highscore	cooperation	Constraints on cooperation
Temporal	No timeout	Semi-timeout	Strict timeout	Constraints on finding posts in time
Mobility	No order	Semi-order	Strict ordering	Constraints on finding posts in order
Goal	FIND	FINDANDSOLVE	SOLVE	Constraints on calculating points

Table E.1: Matrix of dimensions

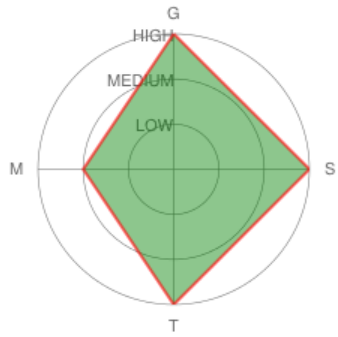


Figure E.1: An example of game, by combining axis

Listing E.1: Grammar Definition for third prototype

```
grammar com.habii.treasurehunt.Treasurehunt with org.jscience.xtext.Jscience

generate treasurehunt "http://www.habii.com/treasurehunt/Treasurehunt"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

TreasureHunt:
    'World: ' name=EString
    elements+=AbstractElements*
    config=Config
;
AbstractElements:
    Post | Player | Team
;
Post:
    'Post' name=EString '{'
        'title: ' title=EString
        'position: ' position=ELatLong
        ('task: ' task=Task)?
        ('clue: ' clue=Clue)?
```

```
        ('visitedBy: ' visitedBy+=[Player|QID] (',' visitedBy+=[Player|QID])*)?
        ('world: ' world=[TreasureHunt])
    '}'
;
Player:
    'Player' name=EString '{'
        'fullname: ' fullname=EString
        'initialClue: ' initialClue=[Clue|QID]
        'distanceToClosestPost: ' distanceToClosestPost=DistanceToClosestPost
        ('visited: ' visited+=[Post|QID] (',' visited+=[Post|QID])*)?
        ('solved: ' solved+=[Task|QID] (',' solved+=[Task|QID])*)?
        ('currentPosition: ' currentPosition=ELatLong)?
        ('direction: ' direction=Direction)?
        ('world: ' world=[TreasureHunt])
    '}'
;
Team:
    'Team ' name=EString '{'
        members+=[Player|QID] (',' members+=[Player|QID])*
    '}'
;
QID:
    EString ('.' EString)*
;
Task:
    name=EString '#' question=EString'?' answer=EString'!'
    ('solvedBy: ' solvedBy+=[Player|QID] (',' solvedBy+=[Player|QID])*)?
;
Clue:
    name=EString '#' text=EString
;
Config:
    'Config' name=EString '{'
        'world: ' world=[TreasureHunt]
        'goal: ' goal=Level
        mobilityAxis=Mobility
        temporalAxis=Temporal
        socialAxis=Social
    '}'
;
Mobility:
    'Mobility' name=EString '{'
        level=Level
        ('ordering: ' ordering+=Order (',' ordering+=Order)*)?
    '}'
;
Order:
    p1=[Post|QID] '->' p2=[Post|QID]
;
Temporal:
    'Temporal' name=EString '{'
        level=Level
        ('deadline: ' deadline=Deadline)?
        ('for: ' posts+=[Post|QID] (',' posts+=[Post|QID])*)?
    '}'
;
Deadline:
```

```
77      days=INT 'd'
78      hours=INT 'h'
79      minutes=INT 'm'
80      seconds=INT 's'
81  ;
82  Social:
83      'Social' name=EString '{'
84          level=Level
85          (teams+=[Team|QID] (',' teams+=[Team|QID])*)?
86      '}'
87  ;
88  enum Level:
89      LOW = 'LOW' | MEDIUM = 'MEDIUM' | HIGH = 'HIGH'
90  ;
91  enum Direction:
92      NORTH = 'W' | SOUTH = 'S' | EAST = 'D' | WEST = 'A'
93  ;
94  enum DistanceToClosestPost:
95      FOUND = 'FOUND' | HOT = 'HOT' | WARM = 'WARM' | LUKEWARM = 'LUKEWARM' | COLD = '
            COLD'
96  ;
97  EString:
98      STRING | ID;
```

Step 2 - Run the generator

This step is the same in all the iteration/prototype, since it just generates artifacts according to the grammar definition (step 1).

E.0.19 Example Game instance for Treasure hunt DSL(v3) .th

The following shows an instance of our third prototype Treasure hunt DSL (.th).

Listing E.2: Treasure hunt DSL (v3) sample script

```
1  World: Advanture
2
3  // Posts
4  Post p1 {
5          title: "Paradise Lake"
6          position: 63.424103,10.40062
7          task: t1# "Hwo named this lake"? "the natives"!
8          clue: c1# "Find the snow..."
9          world: Advanture
10 }
11 Post p2 {
12         title: "Fearsome Forest"
13         position: 63.420723,10.39993
14         task: t2# "How many trees are in this forest"? "1000000"!
15         clue: c2# "Clue to Post4"
16         world: Advanture
17 }
```

```
18 Post p3 {
19        title: "Pine Trees Beach"
20        position: 63.4256386,10.3983879
21        task: t3# "Who comes here often"? "Pirates"!
22        clue: c3# "Where do pirates hang out?..."
23        world: Advanture
24 }
25
26 // Players
27 Player pl1 {
28        fullname: "Habibollah Hosseinpoor"
29        initialClue: Advanture.p1.c1
30        distanceToClosestPost: WARM
31        currentPosition: 63.425076,10.40155
32        direction: A
33        world: Advanture
34 }
35
36 Player pl2 {
37        fullname: "Christian Skar"
38        initialClue: Advanture.p1.c1
39        distanceToClosestPost: COLD
40        currentPosition: 63.4257,10.39619
41        direction: A
42        world: Advanture
43 }
44
45 // Teams
46 //Team teamA {
47 // player1,player2
48 //}
49 //
50 //Team teamB {
51 // player2,player1
52 //}
53
54 Config Simple {
55        goal: HIGH
56
57        Mobility m1 {
58               MEDIUM
59               ordering: Advanture.p1->Advanture.p2 , Advanture.p3->Advanture.p2 //
                      semi-order for MEDIUM strict-order for HIGH
60        }
61
62        Temporal t1 {
63               MEDIUM
64               deadline: 0d 0h 1m 0s // only for MEDIUM and HIGH
65               for: Advanture.p1,Advanture.p3 // only if MEDIUM
66        }
67
68        Social s1 {
69               MEDIUM
70 //  teamA,teamB // only for HIGH
71        }
72 }
```

Step 3 - Updating the engine

This iteration/prototype entailed by far the biggest workload, specially for making the engine to interpret concepts according to the defined semantics as already mentioned. Never the less we managed to implement the engine to interpret the following concepts in this iteration:

Mobility LOW, MEDIUM and HIGH

Temporal LOW, MEDIUM and HIGH

Social LOW, MEDIUM (partly)

Goal LOW, MEDIUM, and HIGH.

As for the Social dimension, we weren't able to implement our engine to interpret level HIGH, due to time constraints.

The app UI

Figure E.2, shows our third engine prototype during the execution of an instances of treasure hunt game. As it can be seen from the Figure E.2, we also removed the buttons for moving the player object around the map, we now can move the player object by keys (A = west, W= north, S=sought, D=east).

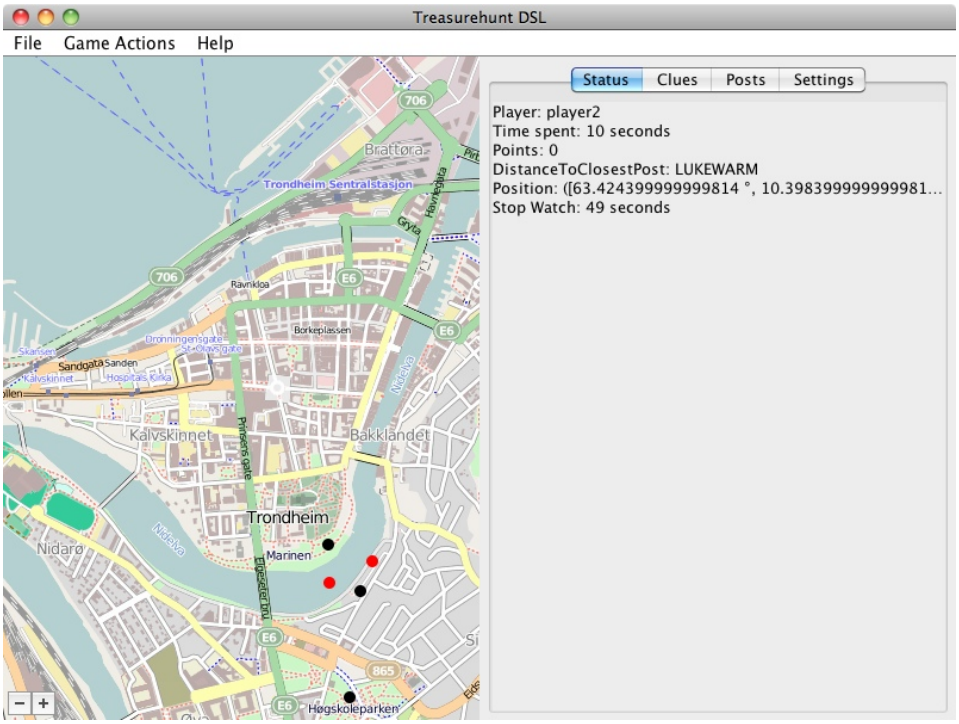


Figure E.2: Treasure hunt third prototype

Appendix F

Fourth Prototype

This appendix presents the results of our fourth iteration/prototype of treasure hunt application.

Introduction

The aim of this prototype was to develop a graphical editor for creation of games as an alternative for using the textual editor generated by Xtext, in order to simplifying the creation of treasure hunt games.

The app UI

Figure F.1, shows the end result of this iteration, namely the graphical editor integrated into the engine, thus simplifying the deployment and improves the usability of the application.

To create a game instance, you need to create a set of post and player objects, and also configure the pervasive dimension (axis, see Chapter 4). Once you have create a post or a player object you can click on the newly created object on map view, which then allow you to edit the selected object, as Figures F.2 and F.3 shows.

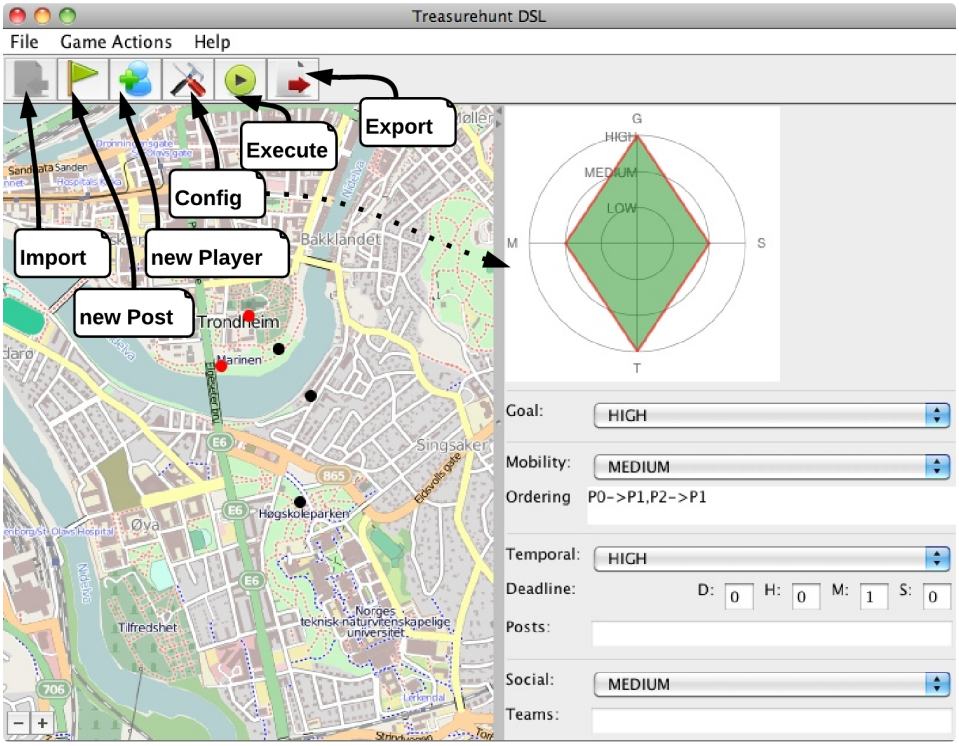


Figure F.1: Graphical Editor - fourth prototype

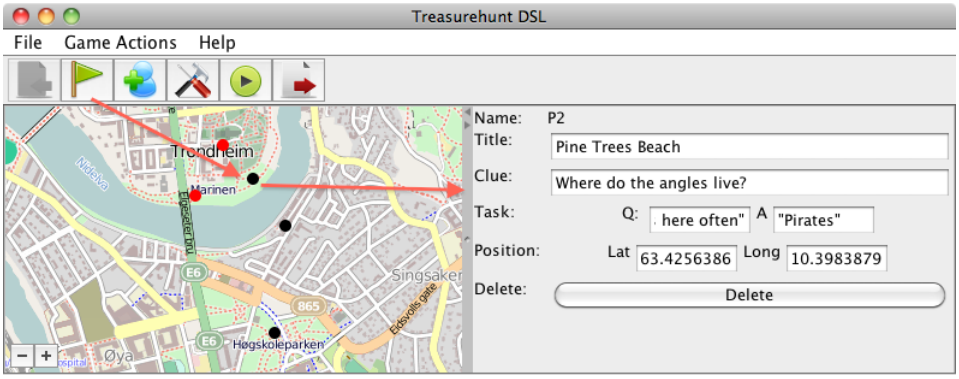


Figure F.2: Graphical Editor - Creating and editing a Post object

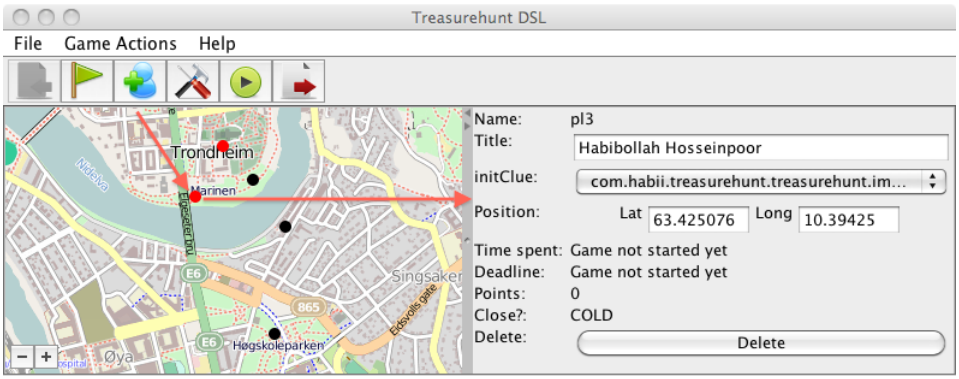


Figure F.3: Graphical Editor - Creating and editing a Player object

Appendix G

User manual

This appendix presents the user manual for our final prototype.

Introduction

Treasure hunt **DSL** is simple and powerful language for creating treasure hunt games, without the need of technical (programming) knowledge and expertise. It comes with an **Engine** with are able to execute the model created in the treasure hunt DSL, as well as a **Graphical Editor** for creating models. How ever one can also create models using a simple text editor.

What is included

Table G.1, shows what is included in the deliverables.

Artifact	Description
TreasureHunt3.2.jar	The last and current prototype of the treasure hunt engine with the graphical editor.
SourceCode	The source code for DSL, Engine and the Graphical editor.
Samples	Game samples written in treasure hunt DSL.
Readme.txt	Software dependencies, legal stuff.

Table G.1: Item Check List

Hardware Specification

This software is tested on Macbook Pro, with Mac OS X operative system version 10.6.8, which comes with Java preinstalled.

We assume that this software works on every operative system with the latest java runtime installed, but we do not guaranty it.

Installation

There is no need for installation. To run the application just double click on the TreasureHunt3.1.jar file, which start both the Engine and the graphical editor.

Basic Operations

G.0.20 Get Familiar with DSL

An example game written in treasure hunt DSL is shown bellow, where the three most important concepts are Post, Player and Config. Each of this concepts contains other concepts, see Table G.2 for more details.

Concepts	Description
World	has an identifier, and contains the following concepts; Post(s), Player(s) and Config
Post	has an identifier and a title. It contains the following concepts; Position, Task, Clue and a reference to the world
Player	has an identifier and a fullname. It contains the following concepts; Position and references to an initial clue, distanceToClosestPost and a reference to the world
Config	has an identifier. It contains the following concepts; Goal, Mobility, Temporal and Social. And a reference to the world
Position	contains latitude and longitude values
Task	has an identifier and contains a pair of question and answer.
Concepts	has an identifier and contains a clue text for the next post.
Order	contains references to posts
Deadline	contains numbers which represents time

Table G.2: Domain concepts

```
1  World: Trondheim
2  Post P0 {
3         title: "Paradise Lake"
4         position: 63.424103,10.40062
5         task: t0 # "How named this lake"? "the natives"!
6         clue: c0 # "Where do the trees speak?"
7         world: Trondheim
8  }
9
10 Post P1 {
11        title: "Fearsome Forest"
12        position: 63.420723,10.39993
13        task: t1 # "How many trees are in this forest"? "1000000"!
```

```
14          clue: c1 # "Where do people go for vacation?"
15          world: Trondheim
16  }
17
18  Post P2 {
19          title: "Pine Trees Beach"
20          position: 63.4256386,10.3983879
21          task: t2 # "Who comes here often"? "Pirates"!
22          clue: c2 # "Where do the angles live?"
23          world: Trondheim
24  }
25
26  Player pl3 {
27          fullname: "Habibollah Hosseinpoor"
28          initialClue: P0.c0
29          distanceToClosestPost: COLD
30          currentPosition: 63.425076,10.394250000000017
31          world: Trondheim
32  }
33
34  Player pl4 {
35          fullname: "Christian Skar"
36          initialClue: P0.c0
37          distanceToClosestPost: COLD
38          currentPosition: 63.42670000000003,10.39619
39          world: Trondheim
40  }
41
42  Config Simple {
43          world: Trondheim
44          goal: HIGH
45
46          Mobility m1 {
47                  MEDIUM
48                  ordering: P0 -> P1 , P2 -> P1
49          }
50
51          Temporal t1 {
52                  HIGH
53                  deadline: 0d 0h 1m 0s
54          }
55
56          Social s1 {
57                  MEDIUM
58          }
59  }
```

G.0.21 Get Familiar with Graphical Editor and Engine

Figure G.1, shows the graphical editor which is integrated together with the engine.

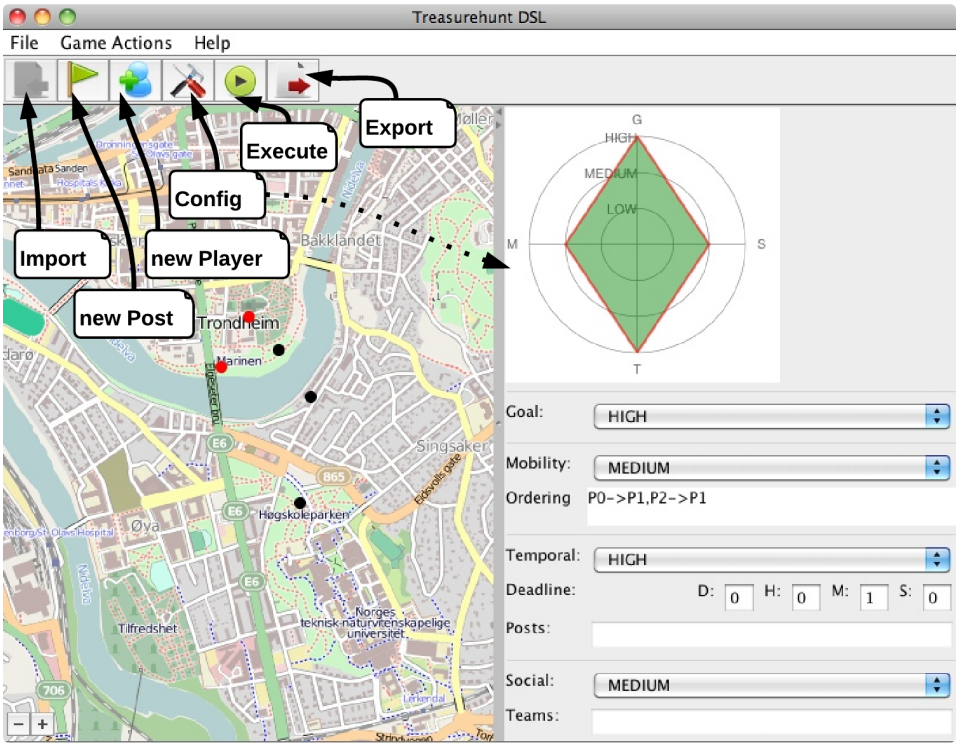


Figure G.1: Graphical Editor

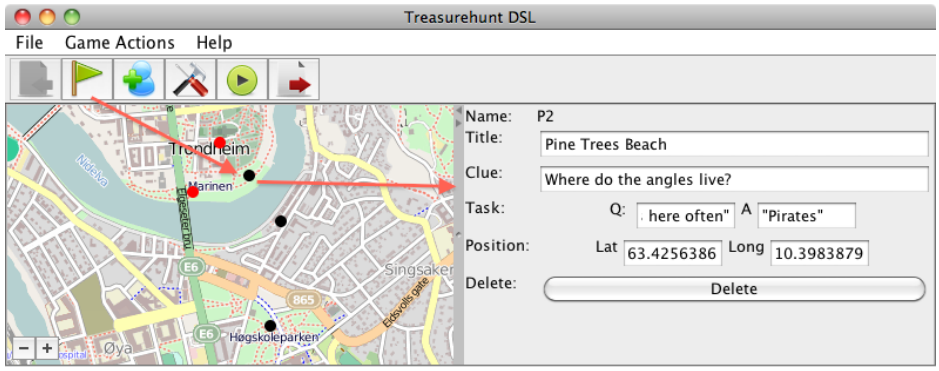
Create treasure hunt games

In order to create a game instance, you have to create a set of post objects, a set of player object, and configure this objects by means of dimensions (axis, see Chapter 4.

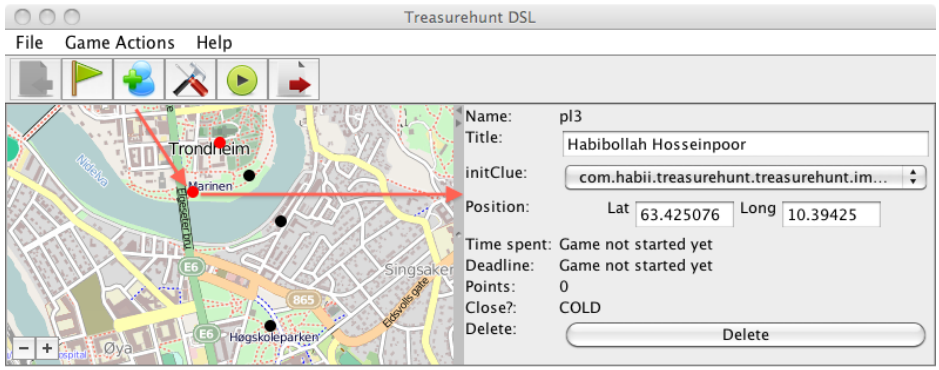
Create Post object Click on the flag icon in order to create a post object. Once the object is created, click on the newly created object on the map view in order to edit its properties, see Figure G.0.21 G.2(a).

Create Player object Click on the blue user icon in order to create a player object. Once the object is created, click on the newly created object on the map view in order to edit its properties, see Figure G.0.21 G.2(b). Player objects can be moved around, by clicking on the following keys (W for north, S for south, A for west. and D for east) on keyboard.

Configure dimensions Click on the Config icon in order to configure the dimensions, see Figure G.1. NB! due to a design decision, textfields changes are effected only when the user hit the **Enter** key on the keyboard.



(a) Creating and editing a Post object



(b) Creating and editing a Player object

Figure G.2: Creating and editing treasure hunt DSL scripts (object) with graphical editor

Notice that you can also create treasure hunt games using a textual editor. For samples of games written in treasure hunt DSL see folder **samples**, which is included in the deliverables.

Run treasure hunt games

By clicking on the **run/play** icon, see Figure G.1, the engine starts executing a game instance/ model. It goes without saying that you need to feed a model to the engine first. This can be done either by importing a game instance into the engine first, or creating a new model using the graphical editor. Figure G.3, shows a complete process as an UML activity diagram.

Once, the game has started, an admin can move players around in the map, by fist selecting a player then clicking on the following keys (W,S,A and D), in order to find post objects.

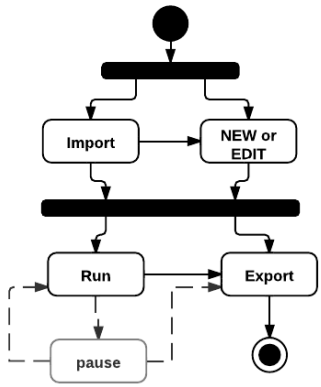


Figure G.3: Activity diagram for creation and execution

List of acronyms

PLAYTRD - is a platform for research about pervasive games. Typical research topics are game concepts, architecture and technology. The purpose of these games is for the players to familiarize themselves with Trondheim by using it as play area. The platform is being developed at IDI in cooperation with Trådløse Trondheim, Telenor and StudentbyEN.

DSL Domain-Specific Language.

GPL Generic Programming Language.

GUI Graphical User Interface.

EMF Eclipse Modeling Framework.

UML Unified Modeling Language is a standardized general-purpose modeling language, it is managed, and was created, by the Object Management Group.

OTS "Off-The-Shelf" component.

COTS Commercial "Off-The-Shelf" component.

HTTP Hypertext Transfer Protocol.

Turing complete Means in principle a turing complete language, could be used to solve any computation problem, although without no guarantees regarding runtime or memory.

IDE Integrated Development Environment.