

# RealCoins: A Case Study of Enhanced Model Driven Development for Pervasive Games

Hong Guo, Hallvard Trætteberg, Alf Inge Wang and Shang Gao

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

*{guohong, hal, alfw, shanggao}@idi.ntnu.no*

## **Abstract**

*Model Driven Development (MDD) and Domain Specific Modeling (DSM) have been widely used in information system domains and achieved success in many open or in-house scenarios. But its application in the game domain is seldom and immature. In our research, we identified three issues that should be considered carefully in order to play the strength of MDD in the game development environment to a larger extent: 1) structured domain analysis should be done to assure the size and familiarity of the domain; 2) adapted process should be designed to save cost and support evolution; and 3) proper tools (especially language workbenches) should be evaluated and utilized to ease DSM tasks and accelerate iterations. In this paper, we explain these three issues and illustrate our solutions to them by presenting the development details (both technical and procedural) of one pervasive game case. We evaluate the gains and costs by involving MDD into the game development process. We reflect on the issues we have met, and discuss possible future works as well.*

**Keywords:** *Model Driven Software Development, Domain Specific Modeling, Pervasive Game, Computer Game, Process.*

## **1. Introduction**

Using models to design complex systems including software systems is not new. Models can help us understand the problem and potential solutions through abstractions. The popularity of UML [1] made even more people understand and accept the importance of model and design by means of model [2]. However, due to practical reasons, models in software engineering were infrequently used. Or, even when used, they often played a secondary role [3]. Until later when Model Driven Development (MDD) and Domain Specific Modeling (DSM) methods were devised to take more advantage from models, models became the primary and only artifacts that needed to be made. MDD is based on two key factors: abstraction and automation. By providing abstractions that are closer to the problem domain, the complex problem domain knowledge is embodied and becomes easier to use [4]. By providing automation, the complexity of solution domain is hidden and full-scale code generation becomes possible. As a result of abstraction and automation, more people without much domain knowledge or programming experiences can write a full specification of the system and generate the software. Further, the productivity, the quality and maintainability of software systems are increased [2].

While MDD has been used widely and successfully in many domains, researchers also tried to apply it in the computer game domain. There are a number of potential advantages of this application. *First*, game software is complex on both the domain knowledge and the architecture. MDD helps to separate them (by hiding the domain

complexity in DSM artifacts). Thus the overall complexity is alleviated. *Second*, it is quite common that game software utilizes a generic game engine which executes specific level descriptions which are produced in a level editor. MDD helps to achieve similar goal: generic code patterns executes specific data (model) which is produced in a Domain Specific Language (DSL) editor, but in a more structural and customized way. Also, the automation degree is expected to be much higher.

However, MDD's application in computer game domains is not quite common and mature as expected till now. During the process of our research on this field, we identified several issues that are quite common or not solved in a desired way which may partly contribute to the current unsuccessful situation. In this paper, we illustrate these issues and present our solutions to them by a case study about the development of the location-based game *RealCoins*. We illustrate the issues in Section 2, Section 3, and Section 4. Then we demonstrate the case study with our solutions to the issues in Section 5. In Section 6, we discuss the costs and gains by involving MDD to our pervasive game development process. We conclude this paper and point out future works in Section 7.

## **2. Structured Domain Analysis to Assure the Effectiveness and Efficiency**

Despite all the benefits DSM may probably bring to software development, DSM is not easy and cheap. Applying DSM does not always sustain its costs. In [5], the author promoted to use DSM whenever it is possible, but still warned that there are some circumstances where DSM solution may not be so plausible. Such circumstances include for example short-term projects or unfamiliar domains. That is why [6] proposed that there should be a decision stage when whether to involve DSM should be decided according to the specific situation among the overall four stages of DSM. The other three stages of DSL development are analysis stage, design stage, and implementation stage. Domain analysis within analysis stage plays an important role because it supports the decision stage with solid data like the domain size for decision-making, and it provides detailed and structured domain knowledge for the design and development. Such knowledge includes: a domain vocabulary with semantic meanings, a model describing the commonality and a model describing the variability space of the domain. These kinds of information are vital for the Domain Specific Language (DSL) meta-model construction that mainly consists of concepts, attributes and relationships. While DSL concepts often come from the domain vocabulary directly, attributes and relationships can be thought as the main means to implement variability (by instantiating and integrating). Such knowledge is also very important for the construction of the generator and the domain specific library [6].

As said above, domain analysis is very important since it ensures the proper size of the domain and the familiarity of participants to decide whether to use DSM. What is more, it provides solid and structured data to ensure an effective and efficient construction of DSM artifacts. However, this part has not been paid enough attention to or, at least has not been done in a visible way among most practices described in the literature regarding to model driven computer game development. Despite various architectures or DSLs they have created or used, few of them present a specified domain definition and a structured domain analysis process to make abstractions.

The work in [7] may be the only one that illustrated the detailed domain analysis process and result structure. In this paper, core dimensions for the game development were considered and analyzed, and the results were recorded in a feature model with almost 150 features (which models the commonality and variability).

## **3. Adapted Process to Save Costs and Support Evolution**

Both computer game development and domain specific modeling are not simple. Increasing challenges due to the overall project size or domain specific characteristics make game development much harder than before during the past decades [8]. But the hardest part of game development has always been the engineering [8]. Computer game development takes big technical risks as well as game design risks due to the fact that you never know whether a gameplay design is really appealing before you can try it out [9]. That is why computer game development emphasizes the importance of prototyping and play-testing [10]. Numerous prototypes are constructed and play-testing is carried out to test the gameplay and the overall user experience in an iterative way before a game can finally be finished. On the other hand, domain specific modeling emphasizes the agility and ability to evolve as well (that usually is realized by an agile or iterative process) [5]. As said by [3], when trying to apply a new technology to an existed production environment, process should also be considered besides the software and environment. The combined process should at least meet the needs of both technologies, and ideally, improve the quality and efficiency of them. When discussing about the application of MDD in game development, how to design the overall process to support agile iterations (as a common requirements coming from both participants) is a must naturally. Further, how to adapt the tasks of original processes – by combining overlapped parts, utilizing deliverables produced by each other efficiently – to make the overall process compact and productive can be crucial to the practicality of the application.

As we said, supporting iterative development and keeping a compact process can be important to the application of MDD in computer game development. There are few articles in the domain of model driven game development. Even fewer mentioned the overall process. While [11] emphasized iterative processes and [12] talked about relationships among tasks of computer game development and domain specific modeling, none of them illustrated in detail and came up with an adapted workflow with combined tasks and iteration support.

#### **4. Language Workbench Tools to Ease DSM Tasks and Accelerate Iterations**

Domain specific modeling is not easy [6]. Developing DSL and tools requires not only comprehensive domain knowledge, but also proficient language development techniques [6]. Few people have both. Domain knowledge can be acquired from various technical documents or domain experts. The complexity and difficulty to carry this process (domain analysis) can be lowered by knowledge engineering techniques. Knowledge capture and representation, as well as ontology development [13] for example can be useful. On the other hand, language workbench tools can alleviate the expertise and efforts needed to develop the language and the tool chain. There is inherently reduced set-up cost with language workbenches for DSM approaches [14]. As what we have found, the usage of language workbench tools in this domain has become visible from around 2008. Workbenches that have been used in this domain include: DiaMeta [14], Microsoft DSL Tools [15, 16], and the Eclipse modeling tools [12, 17, 18]. Besides these, Epsilon, MetaEdit and IntelliJ also provide similar tools. Few people talked about why they chose which workbench tools, and analyze how the workbench tools eased their work and accelerate the process. In [5], the author evaluated several language workbench tools (IDEs). But many of the information there may be out of date now due to the rapid development of these software products, especially the open source tools that are available on the Eclipse platform [19].

Different organizations and software may choose the language bench tools that fit them best according to different criteria like feature set that is provided, price, stability, documents, support, community, and etc. For us, full scale support for language definition, DSL editor, validation, and generator construction is the most important factor

since it decides the costs and worthiness of involving DSM greatly. Another reason we choose to use Eclipse-based tools is that they are free. Further, these tools present good stability and inter-operability. However, from our experiences, lack of documentation and sharp learning curve may be the biggest challenges for more people to use them. We will introduce this with more details about its usage for our case study in the next section.

## 5. Case Study

In this section, we demonstrate our solution to the previous raised issues by a case study. First of all, we introduce the game case. Then we describe how we carried the domain analysis in a structured way basing on a pre-defined ontology. We introduce how we embedded the DSM tasks and made a compact process that in overall still keeps iterative. And at last, we present how we made use of different tools on the Eclipse platform to support DSM definition and usage, in an efficient way.

### 5.1. Game Description

Instead of conventional computer games, a pervasive game is used by use for the case study. Pervasive games have emerged during the last ten years. Such games involve more physical and social elements into the game, and blend game and usual life by providing game experience all the time and everywhere. Well known pervasive games are like “Mobio Threat [20]”, “SupaFly [21]”, “Epidemic Menace [22]”, and “Capture the Flag [23]”. A typical pervasive game has features like location-based, involving physical user interfaces, mobility, and long lasting [24]. Below is the description of our pervasive game case:

*Real Coins is a location-based, mobile version of traditional treasure-hunting games. Several groups of players can participate with mobile devices like tablet PCs or smart phones with them. The mobile devices should be equipped with GPS so that position information of players can be sensed and known by the game. To play the game, all players should be physically at the same place and login to the web based game with their group ID and player ID that all players have agreed upon. When the game starts, several treasure zones (with some hidden virtual coins inside) and some other virtual coins outside are scattered in the game area around where players are. In the main view of the game, a real map with information of the treasure zones, coins, players are presented. Players are not able to see the hidden virtual coins before they or their group members enter the treasure zone where hidden coins locate. The main game play is that players need to move physically to enter a treasure zone (by locating within the zone) or get a virtual coin (by co-locating with the coin). An optional gameplay is to steal coins from other group players by approaching them (locating nearby) and pressing a hot key. The action of stealing always costs a fix amount of coins of the player, while the result coins that the player can steal is randomly calculated. Thus it is possible to lose some coins as a result. When the game ends after some time, the group or player with the most coins wins the game.*

### 5.2. Ontology Based Domain Analysis

In this case, we did domain analysis based on a pre-defined ontology named Pervasive Game Ontology (PerGO) [25] which is part of our previous research results. The PerGO ontology contains two levels of abstractions: higher level abstractions that are common to all computer games and lower level abstractions that are primarily used or often used by pervasive games. These abstractions can be used directly or as a base to derive new concepts in target DSL. All the abstractions are organized in perspectives like ‘Challenge and Action’, ‘Virtual World Element’ and ‘Presentation’. These perspectives are helpful to frame domain analysis as well as the concepts identification in a systematic way. Based on the perspectives and abstractions provided in PerGO, we did domain analysis in three

steps: 1) Go through all the perspectives, identify perspectives that relate to current domain, and record them; 2) Go into each perspective that has been recorded, analyze the common game design, express them as concepts by utilizing existed abstractions in PerGO or deriving new ones basing on PerGO; and 3) Go through the perspectives and concepts identified in step 2, imagine and scope possible variability space basing on the concepts, and express the variability as concept attributes or relationships among concepts. Table 1 presents part of the result of domain analysis (due to limitation of space) as below.

Table 1. **Commonality and Variability of RealCoins**

| <b>PerGO Perspectives</b> | <b>Commonality (Concepts)</b>  | <b>Variability</b>   |
|---------------------------|--|--|
| Challenge and Action      | (Challenge): GetMoreCoin;<br>(Actions): Move, GetTreasure, StealTreasure | Game Duration, Max coin of single win or max coin of total win, Whether to support get treasure action, Whether to support steal treasure action, Cost and value range of steal coin action, |
| World Element             | Map, Location, Treasure, Group, Player                                   | Location number, location size, location position, treasure value, treasure position, treasure number, player position   |
| Presentation              | MainView, GUI, LoginGUI  | Map style for client, map style for server, transparency for group members, transparency for opponents,  |
| Control                   | PhysicalMove, GetTreasureByKey   | Key for GetTreasure, key for StealCoin   |

### 5.3. Compact and Iterative Process

As we mentioned previously, computer game software is traditionally developed and polished in a highly iterative way. Within each iteration, game design and game software development progress alternatively. Thus domain analysis cannot be done in an ideal stage located between requirements collection (game design) and software development. As a result when we developing the case, DSM tasks were interleaved with game development tasks, and the process in overall kept highly iterative. We introduce these tasks in a linear way to make our discussion concise.

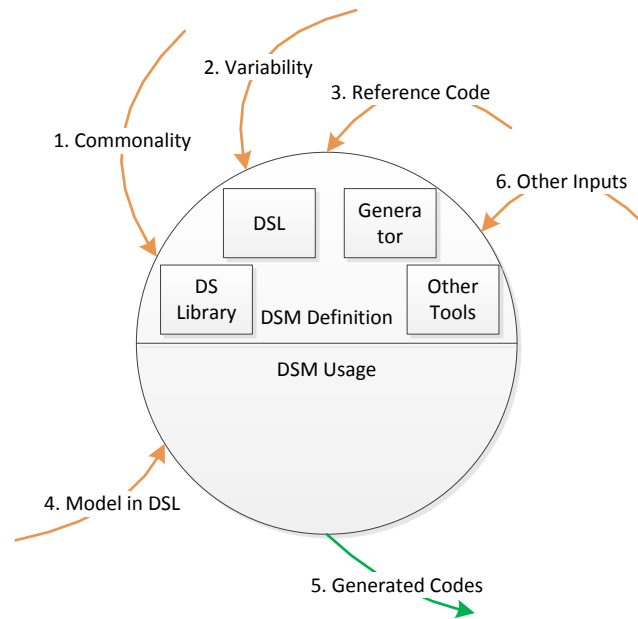


Figure 1. Inputs and Outputs of DSM

As Figure 1 shows, a DSM solution consists of two parts: DSM definition and DSM usage. In the DSM definition, artifacts (DSL, domain specific library for instances) are defined and tools (DSL editor, generator for instances) are produced. While in DSM usage, models are constructed in the DSL editor, then code is generated automatically. DSM hides the domain specific complexity by encapsulating them in DSL concepts and domain specific libraries. Thus the construction of artifacts and tools in DSM definition requires inputs from the domain, usually in the form of Commonality space, Variability space, and Reference Code.

On the other hand, traditional game development mainly involves two stages: Pre-Production stage and Development stage. The main deliverables from a Pre-production stage is Game Design and a workable Baseline Prototype. While the main deliverables from a Development stage is Level Design and numerous Tuning Prototypes to tune the gameplay and the integration of everything. Since game design traditionally decides global settings (common parts among all prototypes) like epoch and space, primary game play, basic game world element types and etc., and level design mainly decides more specific ones (variable parts among all prototypes) like the detailed game play, element objects, and their integration in one level (this stands for one level games as well). Thus we did commonality and variability analysis within these two tasks respectively as Figure 2 shows. Notice that, the variability analysis is done basing on an expectation of possible design for all levels, instead of design for one specific level (which will be mentioned a little bit later). Further, the baseline prototype that has been built can be used as the reference code to build the generator. After DSM language and tools have been defined, the design for each one specific level can be written as a Model in DSL, and the Generated Codes will be available by running the tool chain.

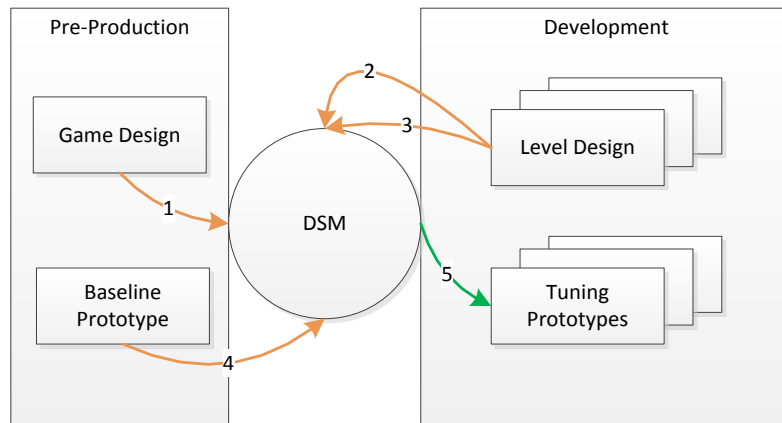


Figure 2. **Game Development Process with Embedded DSM Tasks**

Figure 3 shows UI of the baseline prototype that we have developed (Server side and Client Side). We will illustrate the detailed result of DSM definition and DSM usage together with the usage of language workbench tools in Section 5.4.

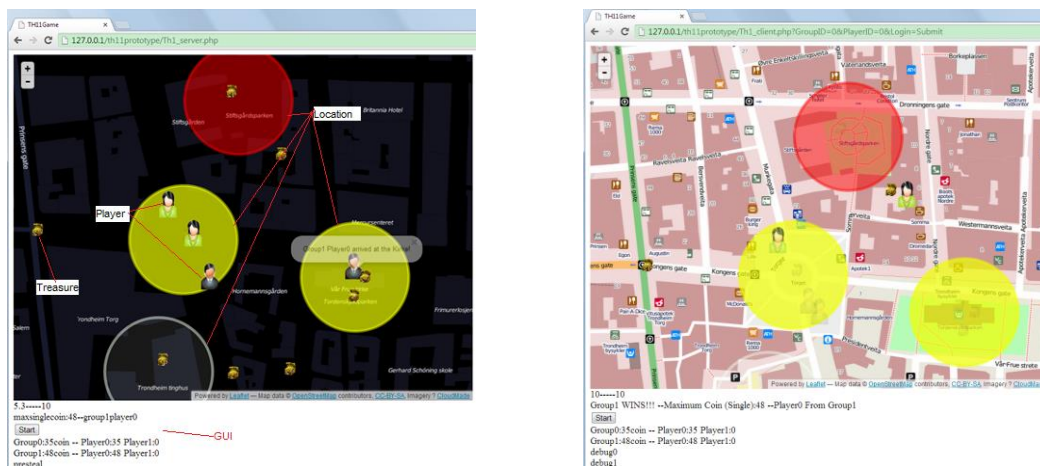


Figure 3. **Server-side and Client-side Applications of RealCoins**

### 5.4 Full Functional Language Workbench

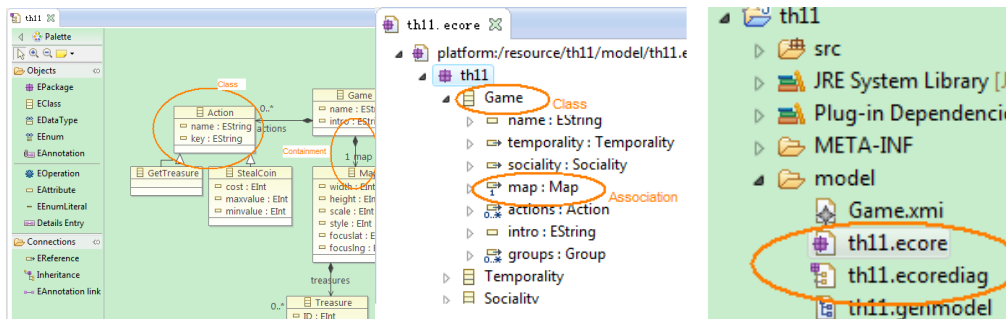
All the tools that we used were based on Eclipse. Eclipse is an open source software community which is available for both individuals and organizations. There are many modeling projects that focus on the promotion and evolution of model based techniques within this community by providing modeling frameworks, tools and standard implementation. Some researchers used different combination of such projects to develop their DSM solutions for computer games [12, 17, 18]. Table 2 is a list of tools that we have used according to which tasks in DSM definition or DSM usage they can support. The tasks are basing on the description in [5]. We will illustrate the details of some of the tools by demonstrating their usage in our case in the left part of this section. To make a complete view, we also listed the domain analysis task (identifying and defining modeling concepts) in the table. It is possible to use any text editor (better to support table) for this task actually. For the integration of multiple languages, we will not introduce in detail since it is not used in this case. Also, maintaining the language requires the cooperation among all the tools instead of a specific one, and will be discussed later. The last two tasks (Domain framework construction and code automation) will not be elaborated also

because they are basing on embedded and implicit mechanisms, and developers do not have to manually write codes or make complex configurations to invoke them usually.

**Table 2. DSM Tasks vs. Eclipse Modeling Tools**

| DSM Tasks [5]   | Tools  |
|---|--|
| Identifying and defining modeling concepts (domain analysis)  | Text editors (can be outside of Eclipse)   |
| Formalizing languages with meta-modeling (defining abstract syntax) by <i>Meta-model Editor</i> (see Section 5.4.1) | “ECore Diagram Editor” (diagrammatic),<br>“Sample Reflective Ecore Model Editor” (tree-based)  |
| Defining language rules by <i>Constraints Validator</i> (see Section 5.4.2)   | “Interactive OCL Console”, testing on “Dynamic Instance” (reference model/ sample model).  |
| Integrating multiple languages  | Importing ECore Model (Not used in this case study)  |
| Notation for the language (defining concrete syntax) by <i>Concrete Syntax Editor</i> (see Section 5.4.3)           | Xtext [26] Editor  |
| Testing the language by creating models in <i>DSL Editor</i> (see Section 5.4.4)                                    | “Sample Reflective Ecore Model Editor” (creating “Dynamic Instance” to test abstract syntax/ meta-model),<br>Generated DSL Editor (to test both met-model and concrete syntax) |
| Maintaining the language  | Cooperation among all the tools (will be discussed later)  |
| Generator definition by <i>Generator Editor</i> (see Section 5.4.5)   | Xtend [27] Editor  |
| Domain framework / domain specific library construction   | Automatically generated GenModel (generating infrastructural classes)  |
| Code automation by generator (see Section 5.4.5)  | Embedded and implicit mechanism  |

**5.4.1 Meta-model Editor:** As presented in Figure 4, there are two editors available to define the abstract syntax of the DSL: a diagrammatic editor (“ECore Diagram Editor” shown in the left part) which allows to draw the meta-model visually, and a tree based editor (“Sample Reflective Ecore Model Editor” shown in the middle part) which allows to specify the concepts and relationship according to the aggregation tree. DSL developers can choose which one to use according to different preference and needs. These two editors save the meta-model in two files separately (shown to the right in Figure 4), but changes to one of them will automatically be applied to another when they are saved. Most of the time, we used the diagrammatic editor because it is more intuitive.



**Figure 4. Meta-model of Sample DSL in ECore Diagram Editor and Sample Reflective Ecore Model Editor**



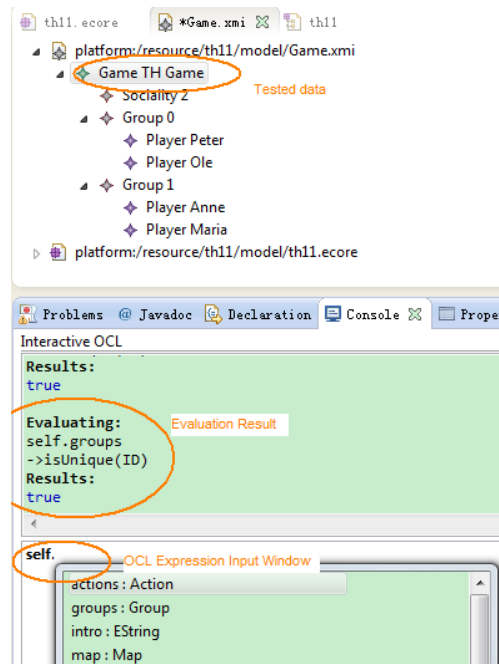


Figure 5. Sample Data and Constraints Validation

**5.4.2 Constraints Validator:** Once the meta-model was decided, a data model (“Dynamic instance” which will be introduced a bit later) can be specified basing on it (without having defined the concrete syntax). Then, we validated constrains which were written in OCL [28]. Figure 5 shows a typical view where OCL constrains were tested on a data model: a node in the data model was selected (upper part of the figure), then the constraints were typed within the console window (lower part of the figure), and the result of the constraint validation was displayed (middle in the figure).



Figure 6. Concrete Syntax Definition

**5.4.3 Concrete Syntax Editor:** When the meta-model has been specified, we chose to automatically generate the concrete syntax of the DSL. In case other language developers intend to invent a set of notations that is more user-friendly, the generated syntax also provides a good starting point which is bug-free and workable. As a result, the overall workflow was accelerated significantly. Figure 6 shows the automatically generated concrete syntax of our DSL in the editor which was enabled by Xtext project [26]. Notice that Xtext supports to define abstract syntax as well actually, but we did not use it to define abstract syntax due to poor visibility of the meta-model.

**5.4.4 DSL Editor:** After defining the syntax, the DSL editor could be generated as a plugin and enabled in a new instance of the Eclipse IDE (also called “Eclipse”). In this new Eclipse instance, we opened a data file written in the DSL, all the keywords would be highlighted, and the data file could be analyzed and understood then. In the left part of Figure 7, it shows that a data file was opened in a common textual editor without any keyword highlighted. In the middle part of Figure 7, it shows that the same file was opened in the DSL editor with all the keywords, numbers, and strings highlighted in different colors (keywords in red, strings in blue, and numbers in gray). In the right part of the figure, we can see that from that time, the data file had been able to be analyzed by the tree-based editor (“Sample Reflective Ecore Model Editor”).

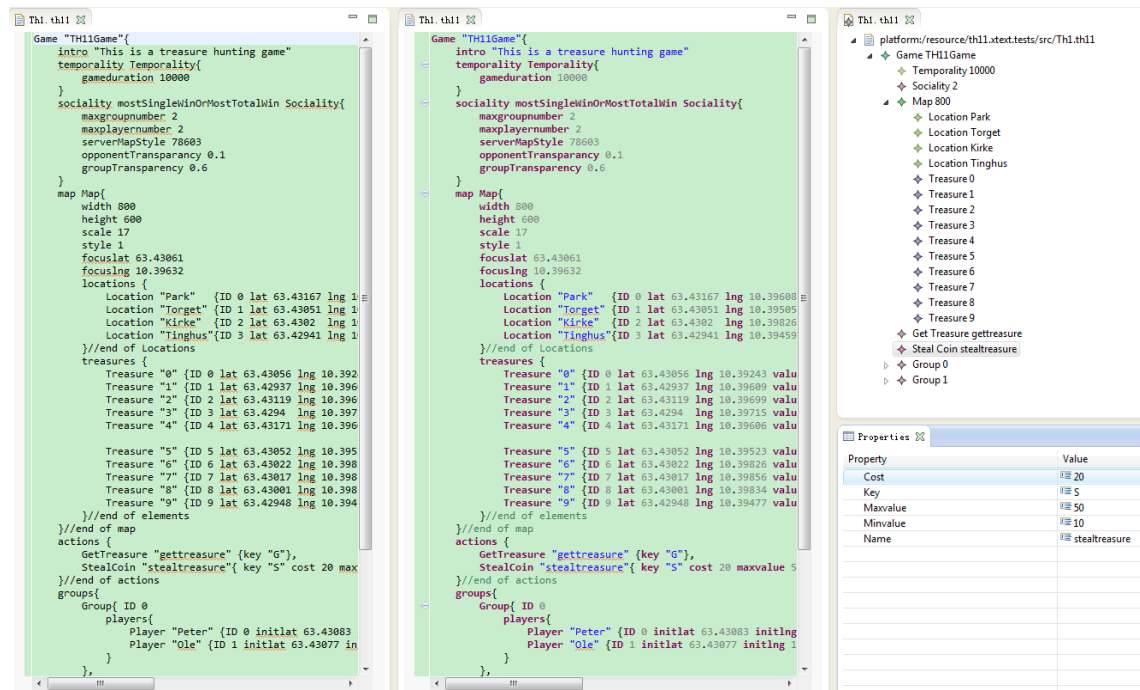


Figure 7. Sample Data in Text Editor, DSL Editor and Sample Reflective Ecore Model Editor

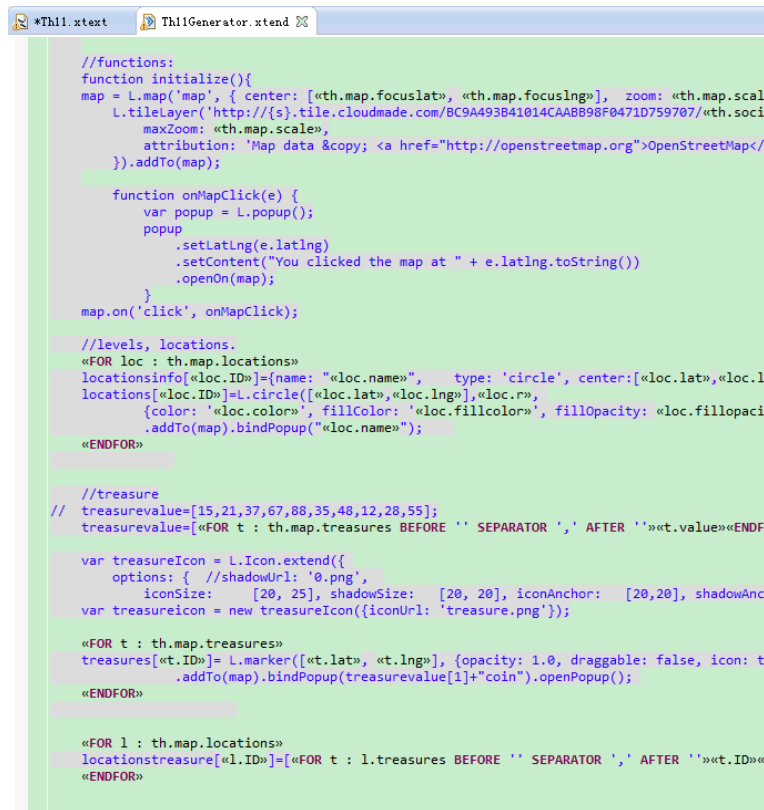
**5.4.5 Generator Editor vs. Generator:** As suggested in [5], a simplified process to construct the generator can utilize the reference codes available (the baseline prototype in our case) by pasting them as the entire content of generated codes, then modify parts that contain repetition or alternatives which relate to the model data. Figure 8 shows a view of the generator editor we used that was enabled by the Xtend project [27]. Figure 9 shows that a set of files (left part) with codes was generated (right part) from the model written in the DSL (middle part).

## 6. Discussion

In Section 5, we followed the three issues we have identified previously and demonstrated how we handled them during the process of developing a MDD solution for location-based treasure hunting games. In this section, we analyze the gain and cost by involving DSM in this case. We reflect on some issues we have met and discussed about possible ways to solve them.

### 6.1 Evaluation

We evaluated the costs of involving DSM to this case by two means: the working hours used and the codes lines written for DSM definition tasks. We evaluated the gains of involving DSM from by these two means also: we estimated hours and codes that can be saved for each prototype we may develop later. In Table 3 and Table 4, we listed the main result. We discuss more about them as below.



```
//functions:
function initialize(){
  map = L.map('map', { center: [«th.map.focuslat», «th.map.focuslng», zoom: «th.map.scale»,
  L.tileLayer('http://s.tile.cloudmade.com/BC9A493841014CAAB898F0471D759707/«th.socia
  maxZoom: «th.map.scale»,
  attribution: 'Map data &copy; <a href="http://openstreetmap.org">OpenStreetMap</a>
  }).addTo(map);

  function onMapClick(e) {
    var popup = L.popup();
    popup
      .setLatLng(e.latlng)
      .setContent("You clicked the map at " + e.latlng.toString())
      .openOn(map);
  }
  map.on('click', onMapClick);

//levels, locations.
«FOR loc : th.map.locations»
locationsinfo[«loc.ID»]={name: "«loc.name»", type: 'circle', center:[«loc.lat»,«loc.l
locations[«loc.ID»]=L.circle([«loc.lat», «loc.lng», «loc.r»,
  {color: ««loc.color», fillColor: '«loc.fillcolor», fillOpacity: «loc.fillopaci
.addTo(map).bindPopup("«loc.name»");
«ENDFOR»

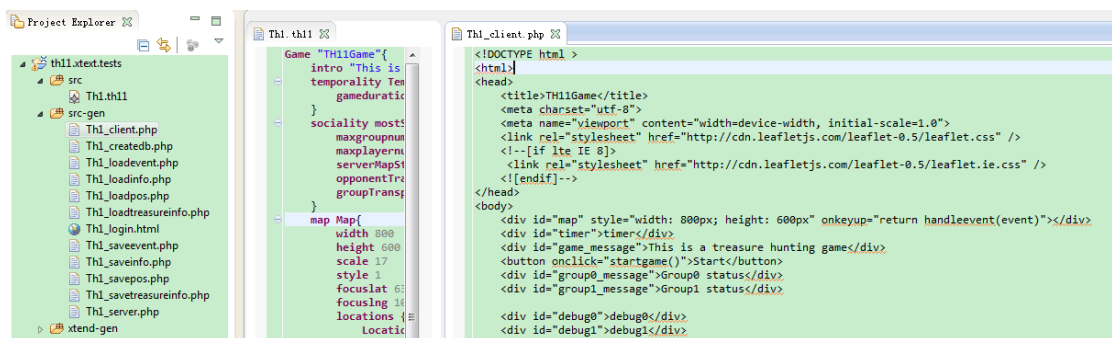
//treasure
treasurevalue=[15,21,37,67,88,35,48,12,28,55];
treasurevalue[«FOR t : th.map.treasures BEFORE '' SEPARATOR ', ' AFTER ''>«t.value»«ENDFOR

var treasureIcon = L.Icon.extend({
  options: { //shadowUrl: '0.png',
    iconSize: [20, 25], shadowSize: [20, 20], iconAnchor: [20,20], shadowAnch
var treasureicon = new treasureIcon({iconUrl: 'treasure.png'});

«FOR t : th.map.treasures»
treasures[«t.ID»]= L.marker([«t.lat», «t.lng», {opacity: 1.0, draggable: false, icon: t
.addTo(map).bindPopup(treasurevalue[1]+«coin").openPopup();
«ENDFOR»

«FOR l : th.map.locations»
locationstreasure[«l.ID»]=[«FOR t : l.treasures BEFORE '' SEPARATOR ', ' AFTER ''>«t.ID»«
«ENDFOR»
```

Figure 8. Generator Definition of Sample DSL



```
Game TH1Game{
  intro "This is
  temporality Tem
  gameduratic
}
sociality mosts
maxgroupnum
maxplayernu
serverMapSt
opponentTri
groupTransj
}
map Map{
  width 800
  height 600
  scale 17
  style 1
  focuslat 63
  focuslng 16
  locations {
  locatic
```

```
<!DOCTYPE html >
<html>
<head>
<title>TH1Game</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.5/leaflet.css" />
<!--[if lte IE 8]
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.5/leaflet.ie.css" />
</endif-->
</head>
<body>
<div id="map" style="width: 800px; height: 600px" onkeyup="return handleevent(event)"></div>
<div id="timer">timer</div>
<div id="game_message">This is a treasure hunting game</div>
<button onclick="startgame()">Start</button>
<div id="group0_message">Group0 status</div>
<div id="group1_message">Group1 status</div>
<div id="debug0">debug0</div>
<div id="debug1">debug1</div>
```

Figure 9. Generated Codes for the Sample Data

In Table 3, it shows that we used 14 hours to develop the first prototype, and 11 hours to develop DSM artifacts. Less than one hour may be needed to develop each one more prototype after that. From this data, it seems that, involving DSM can be worthy when we develop more than 3 prototypes. In Table 4, there were in total 1263 lines of code that have been manually written for the 1<sup>st</sup> prototype, 1232 lines of code for the DSM definition. Developing new prototypes requires around 59 lines for each prototype. The conclusion seems to be in line with the conclusion drawn from Table 3: the gain counteracts the cost from the time when the third prototype is to develop.

**Table 3. Working Hours for Sample DSL Development**

| Tasks                           |                          | Hours                 |
|---------------------------------|--------------------------|-----------------------|
| Constructing baseline prototype |                          | 14h                   |
| DSM definition                  | Meta model and data file | 3h                    |
|                                 | Syntax and generator     | 8h                    |
| Tuning prototypes               |                          | 0.5h-1h per prototype |

**Table 4. Lines of Code for Sample DSL Development**

| Tasks                           |            | Lines of Code (LoC)   |
|---------------------------------|------------|---|
| Constructing baseline prototype |            | 413(client-side) +186(database) +25(login) +419(server-side) +220(database manipulation) = 1263 |
| DSM definition                  | Meta model | 82  |
|                                 | Data file  | 59  |
|                                 | Syntax     | 0(manual) +137(auto-generated)= 137   |
|                                 | Generator  | 173(manual) +1059(pasting from the prototype) = 1232  |
| Tuning prototypes               |            | Around 59 per prototype   |

However, this is not a precise model. In the real circumstances, involving DSM is quite challenging for developers who are used to write application codes in the domain. Even if language developers are involved to develop the DSM solution, the application developers may need more time than us to write codes in the DSL. On the other hand, writing more prototypes that share many common parts do not always cost same hours since the common codes can be pasted and modified. Also, there is reconstruction cost to the DSM solution that is hard to calculate since it relates to the language developer's knowledge and expertise tightly (and also the domain complexity). This cost can be huge at the earlier stage, but will significantly decrease after several iterations (when the DSM gets mature). But anyway, involving DSM can be quite expensive for a project which only aims to several prototypes, but will be more worthy when more prototypes are to be developed (increased productivity).

Besides the increased productivity, from our experiences in this case, we have gained more due to the decreased complexity. Since most of the common domain knowledge has been encapsulated (in DSLs, libraries and generators) and do not need to modify (except for reconstructions), only a few of design details need to be specified in order to produce codes for new prototypes. Obviously the quality and maintainability was improved (as we mentioned earlier) also.

## 6.2 Reflections

In our case, we did domain analysis basing on an ontology that contains common concepts for pervasive games in several perspectives. We felt that perspectives help us structure our analysis well. We felt more confident that we have identified all the concepts

that may be needed in the DSL. On the other hand, many predefined abstractions that can be used directly or with slight modification makes the process much more quicker. Even if we need to make new abstractions, the perspectives and the concepts within them provide us a good context to make such abstractions. Thus in overall, we felt satisfied to the ontology as well the domain analysis process.

However, we also found there may exist places where we can do more. Since the ontology we were using (PerGO) was targeted to various kinds of pervasive game instead of one specific kind (like treasure hunting pervasive game), it is natural that some of the common concepts that are often used in treasure hunting games, like Treasure, are not covered by PerGO since they do not apply to other pervasive games. Further, many attributes and relationships that are common to all treasure hunting games are not covered by PerGO either. This makes that, in case a series of treasure hunting game DSLs are to be developed, such part of domain analysis work is repeated. This raises a question that do we need another layer of artifacts between the ontology and the DSL meta-models, a generic model with all common concepts, attributes, and relationships for various treasure hunting games for example? From engineering's perspective, this may be helpful to develop such DSLs more quickly. However, there are also some open issues. *First*, it is not easy to come up with a clear scope about what elements treasure-hunting games should have. To our knowledge, There are no comprehensive investigation and commonly agreed definitions for such a specific game genre. As a result, developing such a generic model requires a lot of research work to make it generic and useful. *Second*, even if we find a proper way to define treasure-hunting games, the actual games that we need to develop often try to involve different elements that can make them unique, innovative, and appeal. This is caused by the fact that whether a game can succeed mostly depends on its gameplay- whether it is interesting enough- instead of whether it fulfills a pre-defined requirement sets. As a result, it is still inevitable that some domain analysis needs to be done, new concepts need to be identified, and attributes as well as relationships need to be added. At that time, the generic model will need to be reconstructed to be consistent. The reconstruction effort may exceed the efforts that we have expected to save by introducing the generic model. To summarize, involving an extra layer of abstractions brings extra cost and the gain may be unknown in an open discussion. Thus whether it is necessary is an open question, and the answer depends on the specific scenario that developers are facing.

During the overall process, we found that even for such a simple and direct domain, it was still impossible to carry out the tasks one time and everything works fine. We often faced the fact that some artifacts needed to be reconstructed. Such artifacts include reference code (since it cannot fulfill the design), reference model (due to the change of syntax), generator (since it does not generate the code as expected), DSL meta-model (since some of variability cannot be expressed by models that are written in the DSL), level design (since it is not playable), and even game design (since implementing some of the design requires resources that exceed the ability of current platform). However, these artifacts needed to be reconstructed in different frequency. And once an artifact was reconstructed, only the artifacts that were produced in a latter stage in the process needed to be reconstructed accordingly. Normally, the earlier stage one artifact was at (like overall game design), the lower frequency the reconstruction it needed. And by utilizing the automation provided by workbench tools, the time to finish one iteration of reconstruction is shortened to a large extend. From these experiences, we realized more about the importance of an efficient workflow to support evolution. Further, the cost issue for involving DSM, especially about the reconstruction of DSM, must not be overlooked. We should try our best to lower the fixed cost while involving DSL solutions in a specific domain in order to benefit more from it.

With our experience in this case, the language workbench tools available in the Eclipse platform brought us many conveniences. First, the full-spectrum tools make it easy to

build all the artifacts without having to look for other solutions and integrating them. These tools provided many practical functions that saved us a lot of time also. For example, the concrete syntax can be automatically produced which conforms to the abstract syntax. This saves efforts and accelerates the process significantly from our experiences. Another example is, before the concrete syntax is defined, it is possible to create data models that conform to the meta-model to initially validate the meta-model and decreased the possibility to reconstruct the meta-model later greatly. Second, the tools are all open sourced and free, this makes us avoid of cost concern. Third, all the tools we have used in our workflow are presenting high quality and we met few bugs or significant usability issues. However, due to the complexity and lack of documents and other support, we can imagine the sharp learning curve for those who do not have experience on this platform may bring big impact to their choices.

### 6.3 Related Work

There is some related work about applying model driven approaches in game development. However, as model driven software development itself is quite new, model driven game development is even newer. We only found few scientific papers in this domain. While some of them provided higher level envision about this domain [11, 12, 14], some other papers presented concrete demonstrations and initial data as well as lessons they have gained [16, 17, 29]. Very few of them touched in-depth issues like how to perform domain analysis to ensure the quality of the meta-model as it is the most important and starting point in the overall development. Very few of them tried to make some domain-specific adaptation for the process and the tasks. Instead, they mainly treated game development as software development as in common business domains. Thus some special procedural and practical issues could not be considered and took care of. Such issues are like how to combine the traditional game design with model driven tasks, and how to design the overall process so that game design, game software and model artifacts can be developed in an overall iterative way (as both computer game development and model driven approaches require). In addition, we have not found any such paper which focused on pervasive games. We expect our research would provide more insights to this area, especially about the domain analysis strategy and the overall procedure design.

In [30], a conceptual framework was introduced which consisted of three tier design architecture (flow, scenarios and objects) and components (screen components, GUI components, In-game components, and etc.) for serious games. This conceptual framework structured the abstractions and relationships, but might not be able to work as a domain analysis structure in the common way. Also, it was about serious games instead of common computer games. Some concepts were enumerated in the framework which can be thought as a semi-formal vocabulary from our perspective.

[7, 31] may be the work which is closest to our approach. The authors proposed ten dimensions to define a product line including User interface, Game flow, Artificial Intelligence, Sound/Music and etc. A more detailed ontology introduced in [31] was used to define the SharpLudus product line (as the main part of domain analysis task) and create the corresponding DSLs. However, the root concepts in the ontology did not match the ten dimensions of product line definition or the top level DSL concepts in an explicit way. How the product line definition (domain analysis) contributed to the DSL concepts construction based on the ontology precisely is not quite clear.

## 7. Conclusion and Future Work

In this article, we raised three issues about the domain analysis, overall process, and tools usage regarding to the DSM application in game domain. A common focus of these three issues is how to save cost and accelerate process (how to improve the efficiency).

However, it is still an open question about how to evaluate the costs as well as the gains brought by involving DSM. Exploring a more precise model than the traditional comparison among hours used and lines coded can be a useful future work. Further, as our domain analysis relies on a solid and useful ontology largely, we will try to extend, improve, and polish the ontology by more case studies. Also, a more thorough review on available workbench tools can be helpful to our future works as well. For example, Sirius [32] which is available in Eclipse platform as well, supports diagrammatical, tree-based and table-based editing of models, and is promising to provide more visualized and efficient DSM experiences.

## References

- [1] S. Bennett, S. McRobb and R. Farmer, "Object-oriented systems analysis and design using UML", McGraw-Hill Berkshire, UK (2006).
- [2] T. Stahl, M. V. Iter and K. Czarniecki, "Model-driven software development: technology, engineering, management", John Wiley & Sons (2006).
- [3] B. Selic, "The pragmatics of model-driven development", *Software, IEEE*, vol. 20, (2003), pp. 19-25.
- [4] S. Gao and J. Krogstie, "A Combined Framework for Development of Business Process Support Systems", in: A. Persson, J. Stirna (Eds.) *The Practice of Enterprise Modeling*, Springer Berlin Heidelberg (2009), pp. 115-129.
- [5] S. Kelly and J.-P. Tolvanen, "Domain-Specific Modeling Enabling Full Code Generation", John Wiley & Sons, Inc. (2008).
- [6] M. Mernik, J. Heering and A.M. Sloane, "When and how to develop domain-specific languages", *ACM Comput. Surv.*, vol. 37 (2005), pp. 316-344.
- [7] A. Furtado, A. Santos and G. Ramalho, "Streamlining Domain Analysis for Digital Games Product Lines
- [8] *Software Product Lines: Going Beyond*", in: J. Bosch, J. Lee (Eds.), Springer Berlin / Heidelberg (2010), pp. 316-330.
- [9] B. Jonathan, "Game Development: Harder Than You Think", *Queue*, vol. 1, (2004), pp. 28-37.
- [10] R. Rouse III, *Game design: Theory and practice*, Jones & Bartlett Learning (2010).
- [11] T. Fullerton, C. Sawain and S. S. Hoffman, "Game design workshop: designing, prototyping and playtesting games", Taylor & Francis US (2004).
- [12] A. W. B. Furtado, A. L. M. Santos, G. L. Ramalho and E. S. de Almeida, "Improving Digital Game Development with Software Product Lines", *Software, IEEE*, vol. 28, (2011), pp. 30-37.
- [13] R. Walter and M. Masuch, "How to integrate domain-specific languages into the game development process", *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACM, Lisbon, Portugal, (2011), pp. 1-8.
- [14] M. Denny, "Ontology building: A survey of editing tools", *XML.com*, (2002).
- [15] S. Maier and D. Volk, "Facilitating language-oriented game development by the help of language workbenches", *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, ACM, Toronto, Ontario, Canada, (2008), pp. 224-227.
- [16] A. W. Furtado and A. L. Santos, "Using domain-specific modeling towards computer games development industrialization", *The 6th OOPSLA Workshop on Domain-Specific Modeling (DSM06)*, Citeseer, (2006).
- [17] F. E. Hernandez and F. R. Ortega, "Eberos GML2D: a graphical domain-specific language for modeling 2D video games", *Proceedings of the 10th Workshop on Domain-Specific Modeling*, ACM, Reno, Nevada, (2010), pp. 1-1.
- [18] M. Funk and M. Rauterberg, "PULP scription: a DSL for mobile HTML5 game applications, *Entertainment Computing-ICEC 2012*", Springer (2012), pp. 504-510.
- [19] E. Marques, V. Balegas, B.F. Barroca, A. Barisic and V. Amaral, "The RPG DSL: a case study of language engineering using MDD for Generating RPG Games for Mobile Phones", *Proceedings of the 2012 workshop on Domain-specific modeling*, ACM, (2012), pp. 13-18.
- [20] Eclipse, [www.eclipse.org](http://www.eclipse.org).
- [21] W. Segatto, E. Herzer, C. Mazzotti, J. Bittencourt and J. Barbosa, "Mobio threat: A mobile game based on the integration of wireless technologies", *Computers in Entertainment (CIE)*, vol. 6, (2008).
- [22] K. Jegers and M. Wiberg, "Pervasive gaming in the everyday world", *Pervasive Computing, IEEE*, vol. 5, (2006), pp. 78-85.
- [23] I. Lindt, J. Ohlenburg, U. Pankoke-Babatz and S. Ghellal, "A report on the crossmedia game epidemic menace", *Computers in Entertainment (CIE)*, vol. 5, (2007).
- [24] A. D. Cheok, A. Sree Kumar, C. Lei and L. N. Thang, "Capture the flag: mixed-reality social gaming with smart phones", *Pervasive Computing, IEEE*, vol. 5, (2006), pp. 62 - 69.

- [25] G. Hong, H. Trætteberg, A. I. Wang and Z. Meng, "TeMPS: A Conceptual Framework for Pervasive and Social Games", Digital Game and Intelligent Toy Enhanced Learning (DIGITEL), 2010 Third IEEE International Conference on, (2010), pp. 31-37.
- [26] G. Hong, T. Hallvard, W. Alf Inge and G. Shang, "PerGO: An Ontology Towards Model Driven Pervasive Game Development, Ontologies, DataBases, and Applications of Semantics", Springer LNCS, Amantea, Italy, (2014).
- [27] text, <http://eclipse.org/Xtext/>.
- [28] Xtend, <http://www.eclipse.org/xtend/>.
- [29] O. M. Group., Object Constraint Language OMG Available Specification Version 2.0, (2006).
- [30] E. M. Reyno and J. Á. Carsí Cubel, "Automatic prototyping in model-driven game development, Computers in Entertainment (CIE), vol. 7, (2009), p. 29.
- [31] S. Tang, M. Hanneghan, T. Hughes, C. Dennett, S. Cooper, M.A. Sabri, C. Carter, A. El Rhalibi, M. Merabti and P. Fergus, "Towards a Domain Specific Modelling Language for Serious Game Design", 6th International Game Design and Technology Workshop (GDTW'08), Liverpool, UK, (2008), pp. 43-52.
- [32] A. Furtado and A. Santos, "Defining and Using Ontologies as Input for Game Software Factories", Proceedings of the 3rd Brazilian Symposium on Computer Games and Digital Entertainment, (2006).
- [33] Sirius, <http://www.eclipse.org/sirius/>.



## Authors



**Hong Guo**, she is a Ph.D. Research Fellow in Information Systems at the Norwegian University of Science and Technology (NTNU). Her research interests include model driven software development, domain specific modeling, and their application in the computer game domain and the pervasive game domain.



**Hallvard Trætteberg**, he is Associate Professor at the Department of Computer and Information Science at NTNU. His main research interest is model-driven development and in particular model-based development of user interfaces.



**Alf Inge Wang**, he holds a position as Professor in Game Technology at Dept. of Computer and Information Science at the Norwegian University of Science and Technology. He received his PhD degree in 2001, and the main focus of his work is in the intersection of software engineering, ubiquitous computing, education, and game technology. Wang has over 90 international peer-reviewed publications, received the Norwegian Technological achievement of the year award 2014, and is the inventor and a co-founder of the game-based learning platform Kahoot!



**Shang Gao**, he is a Postdoctoral Research Fellow in Information Systems at the Norwegian University of Science and Technology (NTNU). He was Associate Professor at School of Business Administration at the Zhongnan University of Economics and Law, China. He obtained his PhD (2011) in information systems from NTNU, and his MSc (2006) in Engineering and Management of Information Systems from the Royal Institute of Technology (KTH), Sweden. His research interests include mobile information systems, technology diffusion, business process modeling, and information systems modeling. He has published more than 40 refereed papers in journals, books and archival proceedings since 2006.