Concepts and Modelling Techniques for Pervasive and Social Games

Hong, Guo

Doctoral Thesis

Submitted for the Partial Fulfilment of the Requirements for the Degree of

Philosophiae Doctor

Trondheim, June 2015
You will need the two ISBN numbers and the internal NTNU “thesis number” that year. All this can be obtained from http://sats.itea.ntnu.no/isbnprovider/start.do

The ISSN serial number is the same for all doctoral theses at NTNU, and is 1503-8181.
© Hong Guo


ISSN 1503-8181

Thesis at NTNU:2015:144

Printed in Norway by NTNU-trykk, Trondheim
Abstract

Modern computer games are difficult to develop. Emerging pervasive games make this situation even harder, mainly by introducing physical space to computer games. To help developers create games more quickly and easily, various authoring tools are used as they provide visualized user interfaces and code automation. However, these ready-made tools are usually targeted towards certain and narrow game domains. They may not be able to meet certain requirements coming from other game domains. Customized (authoring) tools may provide an alternative between creating games from scratch and creating games with ready-made tools. Model Driven Software Development (MDSD) and Domain Specific Modelling (DSM) can be utilized to implement such customized tools in a formal and efficient way. MDSD has been widely applied in many other domains, and research has shown it to be useful. When it comes to the game domain, research exists where MDSD has been applied. However, some important conceptual and procedural characteristics of computer game development have not been well addressed. These issues can have a major impact on the quality and efficiency of the final application of MDSD. The pursuit of improving the quality and efficiency of model driven pervasive game development has been the inspiration for this Ph.D. thesis. In this thesis, the following research questions are proposed to address the conceptual and procedural challenges:

• **RQ1:** What important concepts need to be considered regarding creating pervasive games with a model driven approach?
  o **RQ1.1:** What important characteristics should/may a pervasive game have?
  o **RQ1.2:** What concepts can be used in a Domain Specific Language (DSL) of pervasive games?

• **RQ2:** How can MDSD techniques be applied in a traditional pervasive/computer game creation process?
  o **RQ2.1:** How can a formalized domain vocabulary be used to enhance the domain analysis process in order to create pervasive games with a DSM approach?
  o **RQ2.2:** How can a traditional computer game development process be adapted to support DSM tasks in an efficient and iterative way?

These research questions are answered mainly through one review and several rounds of DSL development. Case studies and a user acceptance survey were performed to evaluate the research results. The main contributions of this thesis are:

• **RC1:** A conceptual framework named TeMPS (meaning Temporality, Mobility, Perceptibility and Sociality) to summarize important characteristics of pervasive and social games.

• **RC2:** An ontology named PerGO (meaning Pervasive Game Ontology) to structure and accelerate domain analysis for model driven pervasive games development.
- **RC3**: A process named GCCT (meaning Game Creation with Customized Tools) to make use of model driven techniques within the traditional computer game development process.

The first two contributions add to the conceptual base of the computer game and pervasive game domains towards wider application of model driven game development. And the third contribution utilizes the conceptual base, and provides a compact and practical solution to apply MDSD in the computer game domain.
Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfilment of the requirements for the degree of Philosophiae Doctor.

This doctoral work has been performed at the Department of Computer and Information Science, NTNU, Trondheim, Norway under the supervision of Associate Professor Hallvard Trætteberg as the main supervisor, and Professor Alf Inge Wang and Professor Maria Letizia Jaccheri as co-supervisors.

The doctoral work is financed by the Faculty of Information Technology, Mathematics, and Electrical Engineering (IME), NTNU.
Acknowledgements

First, I would like to thank my main supervisor Hallvard Trætteberg, for his supervision, conversations, frankness and patience all the way through my doctoral research. Without his rigorous attitude, my thesis would not have been of its current quality. I would like to thank my co-supervisor Alf Inge Wang for his help and guidance especially in the game domain. He always respected me, inspired me, and supported me in a very constructive way. I would like to express my gratitude to my co-supervisor Maria Letizia Jacceri. I had the chance of receiving her supervision during the darkest days in my life. Her consideration and trust lighted up my hope. Her heuristic guidance and effective support were crucial for my Ph.D.

For the thesis I would like to thank the evaluation committee, Anke Dittmar and Junia Anacleto for the thorough and constructive evaluation. I would also express my thanks to Dag Svanæs for coordinating my Ph.D. defense. I benefited a lot by reading the comments and appreciate all their efforts.

I acknowledge all the researchers whom I have had the opportunity to meet with, to learn from, to exchange idea or work with. I would like to thank my colleagues at IDI and IME for creating a friendly, fair, and enjoyable working atmosphere. Especially I would give thanks to Shang Gao who has given me many valuable suggestions as well as practical help. I would also give my sincere thanks to Bian Wu, Feng Luan, Chengzhi Liu, Reidar Conradi, John Krogstie, Guttorm Sindre, Monica Divitini, Min Shi, Audrius Jurgelionis, Salah Uddin Ahmed, Wei Wei, Jianqiang Ma, Chongsheng Zhang, Huamin Ren, Jingyue Li, Shengtong Zhong, Alfredo Perez Fernandez, Rune Sætre, Tor Stålhane, Herindrasana Ramampilario, Mikhail Fominikh, Nafiseh Shabib, Muhammad Asif, Aparna Vegendla and Anca Deak. I want to thank the administrative staff and technical staff who have been so friendly and dependable helping on the practical issues, especially Anne Berit Dahl, Harald Lenschow, Norun Brembo, Eivind Voldhagen, Knut H. Hassel Nielsen, Kai Torgeir Dragland, and Erik Houmb. Many thanks also would be given to all my friends with whom I could share my troubles and express my happiness. Particularly I would thank Li Zhang, Qiaoduo Shi, Ling Lin, Ye Chen, Jing Xie, Li He, Liyuan Xing, Manman Tian and Chunhong Zhao. I cannot list all their names here, but I treasure all the time I spent with them.

Finally my deepest thanks go to my parents. You always consider about me and support me without conservation. I cannot complete my Ph.D. study without the love between us. Thank my brother, Wei. I clearly know you will give me a hand whenever and wherever it is needed. And to my daughter, Anne, you have brought me infinite happiness and stimulated my biggest potentials. Thank you for choosing me as your mother. In the end I would thank my husband, Meng, for all the delicious food, in-depth discussion, mutual assistance and shared memories which made me a better me.
Hong Guo
June 2015
Contents

Abstract .................................................................................................................. v

Preface .................................................................................................................. vii

Acknowledgements ............................................................................................... ix

Contents ................................................................................................................ xi

List of Figures ...................................................................................................... xv

List of Tables ...................................................................................................... xix

Abbreviations ..................................................................................................... xxi

Chapter 1 Introduction ......................................................................................... 1
  1.1 Motivation 1
  1.2 Problem Outline 2
  1.3 Research Questions 3
  1.4 Contributions 4
  1.5 Research Method 4
  1.6 Publications 8
  1.7 Thesis Structure 9

Chapter 2 Background ....................................................................................... 11
  2.1 Computer Game Concepts 11
    2.1.1 Games ................................................................. 11
    2.1.2 Computer Games .................................................. 12
    2.1.3 Gameplay and Mechanics ..................................... 12
    2.1.4 Vocabularies and Conceptual Frameworks .......... 16
  2.2 Computer Game Development 18
    2.2.1 Phases ................................................................. 18
    2.2.2 Iterations .............................................................. 20
    2.2.3 Game Design and Level Design ............................. 21
    2.2.4 Prototypes ............................................................. 22
    2.2.5 Software Architecture ........................................... 24
    2.2.6 Game Authoring Tools .......................................... 25
    2.2.7 Challenges ............................................................. 30
  2.3 Pervasive Game Concepts and Development 32
    2.3.1 Origins, Elements and Definitions .......................... 32
    2.3.2 Genres and Cases .................................................. 35
    2.3.3 Design and Development ....................................... 37
  2.4 Model and Model Driven Development 38

Chapter 3 Model and Model Driven Development ...................................... 38
  3.1 Model and Model Driven Development ...................................................... 38

Chapter 4 Model and Model Driven Development ...................................... 41
  4.1 Model and Model Driven Development ...................................................... 41

Chapter 5 Model and Model Driven Development ...................................... 43
  5.1 Model and Model Driven Development ...................................................... 43
2.4.1 Model Preliminaries ................................................................. 38
2.4.2 Elements and Products ......................................................... 44
2.4.3 Process, Tools, and Challenges ............................................. 49
2.5 Models for Game Design ......................................................... 55
2.5.1 Models for High Level Game Design ...................................... 55
2.5.2 Models for Low Level and Complex Game Design ................. 57
2.5.3 UML for Game Design .......................................................... 59
2.6 Models for Game Creation ....................................................... 61
2.6.1 Product Line Approaches .................................................... 61
2.6.2 MDA Approaches ............................................................... 63
2.6.3 In-House Approaches .......................................................... 65
2.6.4 Language Oriented Approaches ............................................ 68
2.7 Summary ................................................................................. 72

Chapter 3 Research Method .......................................................... 75
3.1 Challenges .............................................................................. 75
3.2 Research Questions ............................................................... 76
3.3 Research Methods ................................................................. 77
3.4 Research Process ..................................................................... 81

Chapter 4 TeMPS: A Conceptual Framework for Pervasive Games ...... 85
4.1 Motivation ............................................................................... 85
4.2 The TeMPS Conceptual Framework ........................................ 85
4.2.1 Temporality ....................................................................... 86
4.2.2 Mobility ............................................................................. 87
4.2.3 Perceptibility ..................................................................... 87
4.2.4 Sociality ............................................................................ 90
4.3 A Pervasive and Social Games Review by Utilizing TeMPS ......... 90
4.4 Discussion ............................................................................. 93
4.5 Summary .............................................................................. 94

Chapter 5 A Pervasive Game Ontology (PerGO) towards Model Driven
Game Development ........................................................................ 95
5.1 Background ........................................................................... 95
5.2 Pervasive Game Ontology (PerGO) Formalism ....................... 97
5.2.1 Design Decisions ............................................................. 98
5.2.2 Perspectives and the Core Part of PerGO ......................... 100
5.2.3 The Pervasive Part of PerGO ........................................... 101
5.3 A Domain Analysis Procedure Based on PerGO .................... 105
5.4 A Pervasive Game Example .................................................. 106
5.5 Discussion ............................................................................ 108
5.5.1 Review of Domain Analysis Methods in MDGD ............... 109
5.5.2 Towards an Effective and Efficient Domain Analysis Method .... 110
5.5.3 The Costs and Limitations ................................................................. 113
5.6 Summary 113

Chapter 6 Game Creation with Customized Tools (GCCT)................. 115

6.1 Technical Basis 115
6.1.1 A Generalized MDD Workflow .......................................................... 116
6.1.2 Iterative Process ................................................................................. 118
6.1.3 Combined Tasks ............................................................................... 118
6.2 Game Creation with Customized Tools 119
6.2.1 GCCT Formalism ............................................................................. 120
6.2.2 Comparing GCCT and GCEE ............................................................ 123
6.2.3 Selection Guidelines for Game Development Approaches............. 124
6.3 Case Studies 126
6.3.1 Domain Description ........................................................................... 127
6.3.2 Domain Analysis ............................................................................... 128
6.3.3 Process and Tools ............................................................................. 129
6.4 Results 133
6.4.1 Result of Case Studies......................................................................... 134
6.4.2 Cost Analysis from the Domain Perspective ..................................... 136
6.4.3 Cost Analysis from the Practical Perspective ..................................... 138
6.4.4 Lessons learnt for improving productivity ........................................ 140
6.5 Discussion 141
6.5.1 Enhanced Computer Game Development ........................................ 141
6.5.2 Quality as a MDD Approach ............................................................. 143
6.5.3 Towards Minimized Risk and Cost .................................................... 145
6.5.4 Comparison with Other Model Driven Game Development Research 146
6.5.5 Threats to Validity ............................................................................. 147
6.6 Summary 147

Chapter 7 User Acceptance of PerGO and GCCT .............................. 149

7.1 Approach 149
7.2 Settings for the Survey 151
7.3 Demographics of the Respondents 152
7.4 Descriptive Result 153
7.5 Test of Measures 153
7.6 Test of TAM Related Hypotheses (H1- H3) 154
7.6.1 PerGO ............................................................................................. 154
7.6.2 GCCT ............................................................................................. 155
7.7 Test of Additional Hypotheses (H4- H9) 156
7.7.1 Game Domain Knowledge ................................................................. 156
7.7.2 Model Domain Knowledge ................................................................. 158
7.8 Discussion 159
7.8.1 Discussion of Test Results ................................................................. 159
**List of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gameplay Bricks [46]</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>Mechanics, Dynamics, and Aesthetics of a Game [48]</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>User-System-Experience (USE) Model [49]</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Gameplay Mode [34]</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Orthogonal Game Taxonomies [53]</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>Model for Iterative Game Design: Playtest, Evaluate, and Revise [1]</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>Three Stages of the Game Design Process [34]</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>Level Design Process</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>Gameplay Mode [34]</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>Game Software Architecture [62]</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>PlayMaker [77]</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>The GameMaker: Studio Environment [79]</td>
<td>27</td>
</tr>
<tr>
<td>13</td>
<td>RPG Maker XP ENTERBRAIN 2013 [80]</td>
<td>27</td>
</tr>
<tr>
<td>14</td>
<td>Inform7 [81]</td>
<td>28</td>
</tr>
<tr>
<td>15</td>
<td>Articy:draft - Interactive Stories and Dialogues [82]</td>
<td>28</td>
</tr>
<tr>
<td>16</td>
<td>Chat Mapper– Non-Linear Branching Tree Graph Visualization [83]</td>
<td>28</td>
</tr>
<tr>
<td>17</td>
<td>Dash User Interface</td>
<td>29</td>
</tr>
<tr>
<td>18</td>
<td>Kodu Main UI [85]</td>
<td>29</td>
</tr>
<tr>
<td>19</td>
<td>Scratch [86]</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>Video Game Development [87]</td>
<td>31</td>
</tr>
<tr>
<td>21</td>
<td>Occurrence of Game Project Problems [89]</td>
<td>32</td>
</tr>
<tr>
<td>22</td>
<td>Four Axes and the PG Possibility Space [90]</td>
<td>33</td>
</tr>
<tr>
<td>23</td>
<td>Rules, Entities, and Mechanics of PG [90]</td>
<td>34</td>
</tr>
<tr>
<td>24</td>
<td>An Example of Gameflow Design [104]</td>
<td>37</td>
</tr>
<tr>
<td>25</td>
<td>Sample Problem Diagram</td>
<td>39</td>
</tr>
<tr>
<td>26</td>
<td>Sample Feature Diagram [67]</td>
<td>40</td>
</tr>
<tr>
<td>27</td>
<td>Sample Domain Model</td>
<td>41</td>
</tr>
<tr>
<td>28</td>
<td>Modelling and DSLs [118]</td>
<td>45</td>
</tr>
<tr>
<td>29</td>
<td>Aligning Code and Models [17]</td>
<td>46</td>
</tr>
<tr>
<td>30</td>
<td>Model Transformation [118]</td>
<td>47</td>
</tr>
<tr>
<td>31</td>
<td>Platforms in MDD [118]</td>
<td>48</td>
</tr>
<tr>
<td>32</td>
<td>Domain Architecture [118]</td>
<td>48</td>
</tr>
<tr>
<td>33</td>
<td>OMG Model Driven Architecture [110]</td>
<td>52</td>
</tr>
<tr>
<td>34</td>
<td>Why MDA Fails [128]</td>
<td>53</td>
</tr>
<tr>
<td>35</td>
<td>Example of a Flowboard [134]</td>
<td>56</td>
</tr>
<tr>
<td>36</td>
<td>Example of a Story-Beat Diagram [135]</td>
<td>56</td>
</tr>
<tr>
<td>37</td>
<td>Example of Token Interaction Matrix [68]</td>
<td>57</td>
</tr>
<tr>
<td>38</td>
<td>Statechart Diagram for Pacman Game [68]</td>
<td>57</td>
</tr>
<tr>
<td>39</td>
<td>Rhapsody Statecharts of EnemyTracker AI Component [139]</td>
<td>58</td>
</tr>
<tr>
<td>40</td>
<td>Partial Map of Silent Hill 2 Represented in Hypergraph [141]</td>
<td>58</td>
</tr>
<tr>
<td>41</td>
<td>'Game Economy' Diagrams [143]</td>
<td>59</td>
</tr>
<tr>
<td>42</td>
<td>Example of Use Case Diagram and Class Diagram [144]</td>
<td>60</td>
</tr>
<tr>
<td>43</td>
<td>Use Case Diagram of Pacman Game [145]</td>
<td>60</td>
</tr>
<tr>
<td>44</td>
<td>Structure Diagram of Bubble Bobble [147]</td>
<td>60</td>
</tr>
<tr>
<td>45</td>
<td>Example of a Computer Game-Flow Diagram [104]</td>
<td>61</td>
</tr>
</tbody>
</table>
Figure 88. Generator Definition of the Sample DSL.................................................... 134
Figure 89. Generated Codes for the Sample Data ...................................................... 134
Figure 90. Cost Model for a Mixed Approach ........................................................... 145
Figure 91. Technology Acceptance Model [26] ......................................................... 150
Figure 92. Demographics of the Respondents ............................................................ 152
Figure 93. Domain Knowledge of the Respondents ................................................... 153
Figure 94. Structured TAM Model for PerGO ......................................................... 155
Figure 95. Structured TAM Model for GCCT ........................................................... 156
Figure 96. Research Questions and Challenges ....................................................... 164
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Research Questions and Research Contributions</td>
<td>4</td>
</tr>
<tr>
<td>2.</td>
<td>Design Science Research Guidelines [23]</td>
<td>5</td>
</tr>
<tr>
<td>3.</td>
<td>Research Contributions and Research Activities</td>
<td>7</td>
</tr>
<tr>
<td>4.</td>
<td>Gameplay Definitions in Literature</td>
<td>13</td>
</tr>
<tr>
<td>5.</td>
<td>Game Design Process</td>
<td>20</td>
</tr>
<tr>
<td>6.</td>
<td>Selecting Prototyping Methods Based on Game Type [66]</td>
<td>23</td>
</tr>
<tr>
<td>7.</td>
<td>Summary of Location-based Games [102]</td>
<td>36</td>
</tr>
<tr>
<td>8.</td>
<td>Transformability Quality Criteria for Models [131]</td>
<td>55</td>
</tr>
<tr>
<td>10.</td>
<td>Design Science Research Activities [176]</td>
<td>78</td>
</tr>
<tr>
<td>11.</td>
<td>Methodologies in MIS Research [182]</td>
<td>80</td>
</tr>
<tr>
<td>12.</td>
<td>Research Methods Used in the Research of this Thesis</td>
<td>81</td>
</tr>
<tr>
<td>13.</td>
<td>Domain Analysis Table</td>
<td>106</td>
</tr>
<tr>
<td>14.</td>
<td>Domain Analysis Table for the PervasiveTreasureHunting Game</td>
<td>107</td>
</tr>
<tr>
<td>15.</td>
<td>Domain Analysis in the Computer Game Domain</td>
<td>109</td>
</tr>
<tr>
<td>16.</td>
<td>Tasks of Domain Specific Approaches</td>
<td>118</td>
</tr>
<tr>
<td>17.</td>
<td>GCCT as an Example of Model Driven Approaches in the Game Domain</td>
<td>122</td>
</tr>
<tr>
<td>18.</td>
<td>Comparison between GCCT and GCCE</td>
<td>124</td>
</tr>
<tr>
<td>19.</td>
<td>Choosing from Game Creation Approaches: GCAT, GCEE, GCCT and GCS</td>
<td>126</td>
</tr>
<tr>
<td>20.</td>
<td>RealCoins as an Instance of GCCT</td>
<td>128</td>
</tr>
<tr>
<td>21.</td>
<td>Commonality and Variability of RealCoins</td>
<td>129</td>
</tr>
<tr>
<td>22.</td>
<td>DSM Tasks vs. Eclipse Modelling Tools</td>
<td>130</td>
</tr>
<tr>
<td>23.</td>
<td>Hours Used by RealCoins and RealPacman</td>
<td>135</td>
</tr>
<tr>
<td>24.</td>
<td>LoC Used by RealCoins and RealPacman</td>
<td>135</td>
</tr>
<tr>
<td>25.</td>
<td>Detailed Cost of RealPacman</td>
<td>135</td>
</tr>
<tr>
<td>26.</td>
<td>Practical Lessons to Enhance DSM Approaches</td>
<td>142</td>
</tr>
<tr>
<td>27.</td>
<td>Comparing GCCT with Other MDGD Research</td>
<td>146</td>
</tr>
<tr>
<td>28.</td>
<td>Mean Values of TAM Constructs</td>
<td>153</td>
</tr>
<tr>
<td>29.</td>
<td>Internal Consistency of TAM Constructs</td>
<td>154</td>
</tr>
<tr>
<td>30.</td>
<td>Correlation of TAM Constructs for PerGO</td>
<td>154</td>
</tr>
<tr>
<td>31.</td>
<td>Summary of TAM Related Hypotheses for PerGO</td>
<td>155</td>
</tr>
<tr>
<td>32.</td>
<td>Correlation of TAM Constructs for GCCT</td>
<td>156</td>
</tr>
<tr>
<td>33.</td>
<td>Summary of TAM Related Hypotheses for GCCT</td>
<td>156</td>
</tr>
<tr>
<td>34.</td>
<td>Correlations between Pervasive Game Concepts and PU1</td>
<td>157</td>
</tr>
<tr>
<td>35.</td>
<td>Correlations between Pervasive Game Concepts and PEOU1</td>
<td>157</td>
</tr>
<tr>
<td>36.</td>
<td>Mean of PU1 and PEOU1 Corresponding to Different Pervasive Game Knowledge</td>
<td>157</td>
</tr>
<tr>
<td>37.</td>
<td>Correlations between Game Development Experience and PU1</td>
<td>158</td>
</tr>
<tr>
<td>38.</td>
<td>Correlations between Game Development Experience and PU2</td>
<td>158</td>
</tr>
<tr>
<td>39.</td>
<td>Correlations between MDD Knowledge and PEOU1</td>
<td>158</td>
</tr>
<tr>
<td>40.</td>
<td>Correlations between MDD Knowledge and PEOU2</td>
<td>159</td>
</tr>
<tr>
<td>41.</td>
<td>Evaluation of Research Questions</td>
<td>165</td>
</tr>
<tr>
<td>42.</td>
<td>Additional Descriptive Results for PerGO</td>
<td>191</td>
</tr>
<tr>
<td>43.</td>
<td>Additional Descriptive Results for GCCT</td>
<td>191</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>44</td>
<td>Regression for PU1, PEOU1 and IU1</td>
<td>193</td>
</tr>
<tr>
<td>45</td>
<td>Coefficients for PU1, PEOU1 and IU1</td>
<td>193</td>
</tr>
<tr>
<td>46</td>
<td>Regression for PEOU1 and PU1</td>
<td>193</td>
</tr>
<tr>
<td>47</td>
<td>Coefficients for PEOU1 and PU1</td>
<td>193</td>
</tr>
<tr>
<td>48</td>
<td>Regression for PU2, PEOU2 and IU2</td>
<td>195</td>
</tr>
<tr>
<td>49</td>
<td>Coefficients for PU2, PEOU2 and IU2</td>
<td>195</td>
</tr>
<tr>
<td>50</td>
<td>Regression for PEOU2 and PU2</td>
<td>195</td>
</tr>
<tr>
<td>51</td>
<td>Coefficients for PEOU2 and PU2</td>
<td>195</td>
</tr>
</tbody>
</table>
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>Two Dimensional</td>
</tr>
<tr>
<td>3D</td>
<td>Three Dimensional</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>ANTLR</td>
<td>ANother Tool for Language Recognition</td>
</tr>
<tr>
<td>APN</td>
<td>Algebraic Petri-net</td>
</tr>
<tr>
<td>DA</td>
<td>Domain Analysis</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain Specific Language</td>
</tr>
<tr>
<td>DSM</td>
<td>Domain Specific Modelling</td>
</tr>
<tr>
<td>DSML</td>
<td>Domain Specific Modelling Language</td>
</tr>
<tr>
<td>EMF</td>
<td>Eclipse Modelling Framework</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>GCCT</td>
<td>Game Creation with Customized Tools</td>
</tr>
<tr>
<td>GCAT</td>
<td>Game Creation with Authoring Tools</td>
</tr>
<tr>
<td>GCEE</td>
<td>Game Creation with game Engine and level Editor</td>
</tr>
<tr>
<td>GME</td>
<td>Generic Modelling Environment</td>
</tr>
<tr>
<td>GPL</td>
<td>General-purpose Programming Language</td>
</tr>
<tr>
<td>GPML</td>
<td>General-purpose Modelling Language</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HUD</td>
<td>Head-Up Display</td>
</tr>
<tr>
<td>IME</td>
<td>Faculty of Information Technologies, Mathematics, and Electrical Engineering</td>
</tr>
<tr>
<td>IS</td>
<td>Information System</td>
</tr>
<tr>
<td>LARP</td>
<td>Live Action Role-Playing (Game)</td>
</tr>
<tr>
<td>M2M</td>
<td>Model to Model</td>
</tr>
<tr>
<td>M2C</td>
<td>Model to Code</td>
</tr>
<tr>
<td>M2T</td>
<td>Model to Text</td>
</tr>
<tr>
<td>MBSD</td>
<td>Model Based Software Development</td>
</tr>
<tr>
<td>MDA</td>
<td>Model Driven Architecture</td>
</tr>
<tr>
<td>MDD</td>
<td>Model Driven Development</td>
</tr>
<tr>
<td>MDE</td>
<td>Model Driven Engineering</td>
</tr>
<tr>
<td>MDGD</td>
<td>Model Driven Game Development</td>
</tr>
<tr>
<td>MDSD</td>
<td>Model Driven Software Development</td>
</tr>
<tr>
<td>MOF</td>
<td>Meta-Object Facility</td>
</tr>
<tr>
<td>NTNU</td>
<td>Norwegian University of Science and Technology</td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraint Language</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OOFM</td>
<td>Object Oriented Feature Modelling</td>
</tr>
<tr>
<td>PerGO</td>
<td>Pervasive Game Ontology</td>
</tr>
<tr>
<td>PDM</td>
<td>Platform Description Model</td>
</tr>
<tr>
<td>PG</td>
<td>Pervasive Game</td>
</tr>
<tr>
<td>PIM</td>
<td>Platform Independent Model</td>
</tr>
<tr>
<td>PSM</td>
<td>Platform Specific Model</td>
</tr>
<tr>
<td>RA</td>
<td>Research Activity</td>
</tr>
<tr>
<td>RC</td>
<td>Research Contribution</td>
</tr>
<tr>
<td>RQ</td>
<td>Research Question</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>RPG</td>
<td>Role-Playing Game</td>
</tr>
<tr>
<td>RUP</td>
<td>Rational Unified Process</td>
</tr>
<tr>
<td>TAM</td>
<td>Technology Acceptance Model</td>
</tr>
<tr>
<td>TeMPS</td>
<td>Temporality, Mobility, Perceptibility, and Sociality</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>VDSL</td>
<td>Visual Domain Specific Language</td>
</tr>
</tbody>
</table>
Chapter 1 Introduction

In this chapter, the motivation for the research and the problem outline are first introduced. Next, the research questions are presented. Later, the research contributions and their relationships with the research questions are introduced. The research method used is discussed. Some of the related publications are listed and, lastly, the structure for the remainder of this thesis is presented.

1.1 Motivation

Since Spacewar, which is often credited as the first digital game [1], was programmed in 1962 at Massachusetts Institute of Technology (MIT) by Steve Russell and several other students, the game industry has grown to be larger in revenue than traditional entertainment industries like the film industry, and continues to grow quickly. Online games attract millions of gamers, partly because of real-time (it happens now, here) and social (you communicate, play and cooperate with other gamers) aspects, and partly because of the qualities of such games. The degree of innovation in games tends to exceed that of productivity applications. Pervasiveness is one popular type of innovation in recent years. Pervasive games bring more physical movement and social interactions into the game world, and may happen at any time and everywhere in the real world. In today’s mobile world, there is increased interest in pervasive games. Such games depend on where the gamer is and what people and artefacts the gamer interacts with, i.e. they are intrinsically mobile, social and location-aware.

As games become more and more complex, game development faces more challenges. As with other software, an increasingly important attribute of modern game development is variability. Software variability is the ability to change or customize a software system [2]. For computer games, the variability is a direct consequence of the game domain diversity, working with simulations (sports, adventure, fighting), hardware technologies (mobile games, web games), human interactions (immersion, multiplayer) and complex stories (games based on movies, Role-Playing Game (RPG) series) [3]. In [4], variability is identified based on the concept of feature - a logical unit of behaviour that corresponds to a set of functional and quality requirements in a system. To manage the game development variability, due to the game domain diversity, the most successful approach was first achieved by using game engines [4]. As stated in [5], there are currently three levels of programming in games: script code, gameplay code, and engine code. Script and gameplay code control the overall content, rules, and high-level behaviour of the game. By separating these three levels of codes, game assets are portable between games. The games themselves are, however, dependent on the game engine they were developed on. Some types of middleware have attempted to address this by, for instance, separating out the Artificial Intelligence (AI) from the core game engine [6]. In [6], this was developed further by separating elements of the game from the game engine, and making them portable be-
tween game engines. The game elements that were made portable are the game logic, the object model, and the game state, which represent the game’s brain and are collectively referred to as the game factor, or G-factor. This was achieved by using an architecture based around a service-oriented approach. There are additional approaches which aim to manage the variability of games like [7] by supporting game design using a set of game design patterns. However, these attempts have had varying levels of success, and are neither formal nor mature enough to be industrialized.

With some other domains, like the Information System (IS) domain, various modelling techniques, especially Model Driven (Software) Development (MDSD/ MDD), are becoming widely used to lower the software development complexity and shorten the development cycle. Simply speaking, MDSD is the notion that we can construct a model of a system and then transform it into the real thing [8]. Empirical investigations provide evidence to the fact that MDSD may help to meet challenges like increased complexity of products, shortened development cycles, and heightened expectations of quality [9]. MDSD needs to be domain specific, thus Domain Specific Modelling (DSM) is often used in MDSD contexts.

As DSM has been successfully applied in many domains, researchers have tried to apply it in other domains including the computer games domain [10-15]. The computer games domain seems to be desirable for this application due to its complexity and traditions. Computer games can be extremely complex because of their complicated architecture and complex domain knowledge. Applying DSM can potentially alleviate the overall complexity by separating and hiding the domain (knowledge) complexity in Domain Specific Language (DSL). Furthermore, it is quite common for game software to utilize a generic game engine which executes specific level descriptions that are produced in a level editor. DSM proposes a similar approach: utilizing generic code patterns which execute specific data (model) that is produced in a DSL editor, but in a more structural way. DSM applies as well to games without a fully functional game engine or level editors. Also, since the DSL editor can be automatically generated (with meta-model as the only input) by utilizing language workbenches, the overall automation degree is expected to be high.

However, as indicated by [16], DSM/DSL is not used as popularly as in the game development field. The development of DSM is not easy and the application of DSM in computer game domains has not been common and mature. It is even difficult to find scientific papers describing extensive researching of applying DSM in computer game domains (this will be discussed more in Chapter 2).

1.2 Problem Outline

In order to make good use of MDSD approaches in the game domain, scientific papers were reviewed in which the authors illustrate their trials to use model techniques for their game design/development activities. It was found that:

- There are few such scientific papers
- Some trials used Generic-purpose Programming Language (GPL), instead of DSL, although there are recent indications that GPL used for MDSD is not desirable [17].
- Some trials aimed to visualize the concepts (for clarification mostly), instead of improving the development process (using models as the primary artefacts and directing the development process with code generation mechanisms).
Thus they are model based, instead of model driven.

- All of these trials are about traditional computer/video games, and none are about pervasive games.
- Most of the MDSD trials did not show consideration of the traditional game creation process.
- Most of the MDSD trials did not illustrate how the DSL concepts were constructed.

After this review, the research focus for this thesis became clear. It will explore how to apply MDSD approaches as this supports not only the visualization of concepts, but the improvement of the overall process. Special attention should be paid to pervasive games, in addition to common computer games. The thesis will focus on two specific issues for the application of MDSD: 1) how to define the DSL, as it is the first and primary artefact necessary to create in MDSD approaches, and 2) how to carry out MDSD tasks in the overall game creation process in an iterative and efficient way. These issues directly lead to the research questions, which will be introduced in the next section.

1.3 Research Questions

MDSD imposes structure and common vocabularies so that artefacts are useful for their main purpose in their particular stage in the life cycle (such as describing an architecture), for the underlying need to link with related artefacts (earlier or later in the life cycle), and to serve as a communication medium between participants in the project (over space or time) [9]. However, as pointed out in [18], game designers and scholars have discussed the need for a game design language, noting that there is no unified vocabulary for describing existing games and thinking through the design of new ones. While some semi-formal analysis languages are being developed [19], game design (more specifically pervasive game design) has not been described at the level of detail and formality necessary to support automatic generation. Therefore, one important objective of this research will be to identify important concepts in the pervasive game field, and another is to further investigate how to use them in an overall workflow (as introduced in section 1.2). Whether or not the research is useful will also need to be evaluated. Thus the Research Questions (RQs) are summarized as below:

- **RQ1**: What important concepts need to be considered regarding creating pervasive games with a model driven approach?
  - **RQ1.1**: What important characteristics should/may a pervasive game have?
  - **RQ1.2**: What concepts can be used in a Domain Specific Language (DSL) of pervasive games?

- **RQ2**: How can MDSD techniques be applied in a traditional pervasive/computer game creation process?
  - **RQ2.1**: How can a formalized domain vocabulary be used to enhance the domain analysis process in order to create pervasive games with a DSM approach?
  - **RQ2.2**: How can a traditional computer game development process be adapted to support DSM tasks in an efficient and iterative way?

There are two things that need to be clarified regarding the research questions:
Firstly, regarding the term “Pervasive Game”, this will be discussed in detail in Chapter 2. Simply speaking, it refers to one kind of computer game which involves more physical elements and social elements in the games (thus pure physical games, single games, etc. are excluded from the research scope). “Pervasive and Social Games” are also used interchangeably in this thesis since the social aspect is very important.

Secondly, “Game creation” consists of three main tasks: game (and level) design, game software design, and game software development. A distinction is made between the following two design tasks: game design refers to all designs related to game mechanics design, game control design, game presentation design, User Interface (UI) design, hardware design, etc. Such design is used by all game creation participants, including not only software engineers, but also artists, game/level designers, etc. Game software design refers to the design that is specifically useful for software engineers, like software stack design, database design, communication protocols design, software architecture design, etc. In the game industry, people also invest greatly in the game design work, turning marketing or emotional expectations into solid ideas. This part of design will not be covered in this thesis. The primary considerations will be game description and development (something fixed and specific), rather than game invention (something creative, illusive, and variable). The focus is on pervasiveness because it is new [20] and discloses new ways to construct computer games [21, 22] and the corresponding software systems.

1.4 Contributions
As discussed earlier, this research covers both the theoretical part (due to the shortage of available commonly agreed upon theories) and the technical part (due to the necessity of validating the theories). The main Research Contributions (RCs) are:

- RC1: A conceptual framework named TeMPS (meaning Temporality, Mobility, Perceptibility and Sociality) to summarize important characteristics of pervasive and social games.
- RC2: An ontology named PerGO (meaning Pervasive Game Ontology) to structure and accelerate domain analysis for model driven pervasive games development.
- RC3: A process named GCCT (meaning Game Creation with Customized Tools) to make use of model driven techniques within the traditional computer game development process.

These research contributions, as presented in this thesis, answered the research questions listed in section 1.3. Table 1 shows the relationships between them.

<table>
<thead>
<tr>
<th>Table 1. Research Questions and Research Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1.1</td>
</tr>
<tr>
<td>RC1</td>
</tr>
<tr>
<td>RC2</td>
</tr>
<tr>
<td>RC3</td>
</tr>
</tbody>
</table>

1.5 Research Method
Design science [23] is used as the main research method. There are two main paradigms in the research of IS discipline: behavioural science and design science. The
behavioural science paradigm seeks to develop and justify theories that explain or predict human or organizational behaviour. In contrast, the design science paradigm is more problem-solving and seeks to create artefacts to extend the boundaries of human or organizational capabilities. Design is both a process (set of activities) and a product (artefact) [24], and as identified by [25], there are two design processes (build and evaluate) and four design artefacts (constructs, models, methods, and instantiations) produced by design science research in IS. In [23], the authors proposed seven guidelines regarding how to conduct, evaluate, and present design-science research, as Table 2 shows. These guidelines will be discussed below briefly, as to how they are applied to the research of this thesis.

Table 2. Design Science Research Guidelines [23]

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guideline1: Design as an Artefact</td>
<td>Design-Science research must produce a viable artefact in the form of a construct, a model, a method, or an instantiation.</td>
</tr>
<tr>
<td>Guideline2: Problem Relevance</td>
<td>The objective of design-science research is to develop technology-based solutions to important and relevant business problems.</td>
</tr>
<tr>
<td>Guideline3: Design Evaluation</td>
<td>The utility, quality and efficacy of a design artefact must be rigorously demonstrated via well-executed evaluation methods.</td>
</tr>
<tr>
<td>Guideline4: Research Contributions</td>
<td>Effective design-science research must provide clear and verifiable contributions in the areas of the design artefacts, design foundations, and/or design methodologies.</td>
</tr>
<tr>
<td>Guideline5: Research Rigor</td>
<td>Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artefact.</td>
</tr>
<tr>
<td>Guideline6: Design as a Search Process</td>
<td>The search for an effective artefact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.</td>
</tr>
<tr>
<td>Guideline7: Communications of Research</td>
<td>Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.</td>
</tr>
</tbody>
</table>

Guideline1: Design as an Artefact. The three main artefacts developed in this thesis are a conceptual framework (TeMPS) which summarizes important characteristics of pervasive games, an ontology named PerGO which consists of around 100 concepts that are primarily or often used by pervasive games, and a workflow for the overall model driven game creation process named GCCT.

Guideline2: Problem Relevance. Applying MDSD techniques in game development has neither been mature nor successful until now. Common vocabularies are needed to form the basis for automation. How to integrate MDD tasks in the game development process in an iterative and efficient way should be considered further in order to achieve the further application of MDGD. By proposing the conceptual framework TeMPS and the ontology PerGO, one of the aims of this research is to create a common understanding as to what pervasive games are and what is needed by pervasive games creation with MDD approaches. A further aim is to structure and accelerate the
DSL concepts creation. In addition, by proposing the GCCT workflow, another aim of this research is to solve the procedural issue (how to integrate the MDSD tasks in the game creation process in an iterative and efficient way).

**Guideline3: Design Evaluation.** The utility, quality and efficacy of a design artefact must be rigorously demonstrated via well-executed evaluation methods. The conceptual framework TeMPS was validated with a literature review for more than 30 pervasive games, to ensure the framework covered most important characteristics. The internal consistency between TeMPS (containing pervasiveness characters) and PerGO (containing concepts for game construction) was validated by mapping the concepts between them. The usage of PerGO was demonstrated and evaluated in an analytical way. The effectiveness and efficiency of the GCCT workflow was evaluated by developing DSLs for small pervasive game families (RealCoins, RealPacman, etc.), and then comparing the code lines as well as working hours used with this approach versus those used with the traditional manual way. Finally, the user acceptance for PerGO and GCCT was evaluated by performing a survey among researchers and students, based on the Technology Acceptance Model (TAM) [26].

**Guideline4: Research Contributions.** Effective design-science research must provide clear and verifiable contributions in the areas of the design artefact, design foundations, and/or design methodologies. As introduced in section 1.4, this research has provided three contributions (design artefacts) which answered the research questions raised previously.

**Guideline5: Research Rigor.** Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artefact. To construct TeMPS, approximately 30 pervasive games were reviewed. To construct PerGO, a literature review was carried out to make sure that all of the important pervasive games characteristics were clear, and important conceptual frameworks of computer games were examined to ensure the compatibility and extendability of the ontology with common computer games. Clear criteria were also defined to ensure that all the included concepts are in a proper abstraction level with the aim to be used for model driven approaches. To construct the GCCT workflow, both the MDSD process and the traditional computer/pervasive game creation process were studied. After becoming familiar with the requirements and features of these, the available Model Driven Game Development (MDGD) approaches were reviewed and a workflow which fits MDSD tasks in the game creation process in an efficient way was proposed. Threats to validity are discussed in detail in Chapter 8.

**Guideline6: Design as a Search Process.** The search for an effective artefact requires utilizing available means to reach desired ends while satisfying laws in the problem environment. The artefacts were developed in an iterative way. For instance, when the TeMPS framework was developed, a rough framework was first set up, and then refined as articles were reviewed. Also, the development of DSLs was done in an iterative way. MDSD methods are intrinsically iterative. For each iteration, a meta-model was first made, then a data model was developed for it, a prototype created and the generator template filled in. For each step, one or more issues might be found that needed to be fixed. These issues were fixed, the corrections used in the next iteration, and the process continued. PerGO and GCCT were refined in such iterations according to the practical findings.
Guideline 7: Communications of Research. Design-science research must be presented effectively to both technology-oriented as well as management-oriented audiences. Sufficient details must be provided in order to enable described artefacts to be constructed by technology-oriented audiences. Knowledge must also be stated clearly in order to create an effective application of the artefact for management-oriented audiences. This was ensured by discussion and cooperation with other researchers, peer review by conference committees and journal editors, and paper presentations at international conferences. Also, when the user acceptance survey was conducted to evaluate user acceptance, the survey was sent out to not only game researchers and MDD researchers, but also other potential users without game or MDD background knowledge.

Table 3. Research Contributions and Research Activities

<table>
<thead>
<tr>
<th></th>
<th>RC1</th>
<th>RC2</th>
<th>RC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA2</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RA3</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RA4</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RA5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA6</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RA7</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The Research Activities (RAs) carried out during this research are summarized below. The relationship between these research activities and the research contributions is presented in Table 3.

- RA1: Construct the TeMPS framework by reading literature (regarding various pervasive game cases and theories), analysing common characteristics and creating synthesis.
- RA2: Validate TeMPS by reviewing pervasive game literature and fitting corresponding pervasive games into the framework.
- RA3: Construct the PerGO framework by reading literature (regarding computer game conceptual frameworks and computer/pervasive game design), refining and deriving (from TeMPS) concepts and formalizing them in diagrams.
- RA4: Evaluate PerGO by performing domain analysis for an example game and discussing this analytically.
- RA5: Construct the GCCT workflow by reading literature (MDSD, computer game development, and MDGD) and practicing MDSD techniques on the Eclipse platform [27].
- RA6: Evaluate the effectiveness and efficiency of the GCCT workflow by constructing pervasive game DSLs (RealCoins, RealPacman, etc.) based on MDSD and PerGO, discussing subjective experiences, and analysing working hours and code lines.
- RA7: Evaluate the potential user acceptance degree for PerGO and GCCT by performing a survey based on TAM.
1.6 Publications

Parts of this research work were published in the following papers:


- Publication 8, Hong Guo, Hallvard Trætteberg and Shang Gao. "Efficient Game Development through an Adapted Model Driven Approach." *Software and Systems Modeling* (Submitted for review).


- Publication 11, Hong Guo, Hallvard Trætteberg, Alf Inge Wang and Shang Guo.


1.7 Thesis Structure
The rest of the thesis is organized as below.

Chapter 2 presents basic game concepts, game development traditions, as well as state-of-the-art research in the pervasive game domain. Models and model driven techniques are introduced. Research regarding the application of model techniques in game design and game development is also reviewed.

Chapter 3 summarizes the challenges found in Chapter 2. It describes how the research questions were derived from the challenges, and what the research methods and activities used in this thesis are.

Four chapters are used to illustrate the contributions of the research in this thesis.

Chapter 4 introduces the conceptual framework TeMPS and presents the result of the literature review for approximately 30 pervasive games based on the framework. In Chapter 5, the pervasive game ontology PerGO is presented. This chapter illustrates how PerGO looks and how it can be used to structure and accelerate parts of the tasks of DSM. The usage of PerGO is demonstrated by an example, and its quality evaluated. Then, in Chapter 6, a procedural GCCT workflow is proposed to integrate MDSD tasks in the traditional game development process. Its usage is demonstrated and the cost evaluated, as well as the productivity increase resulting from the model driven approach. Later, in Chapter 7, a survey is presented regarding user acceptance of both PerGO and the overall GCCT approach.

In Chapter 8, the research of this thesis is evaluated overall, future works are presented, and the thesis is concluded.

Appendix A-D includes more information about the survey (as introduced in Chapter 7) and its results. Appendix E includes the references.
Chapter 2 Background

In this chapter, background knowledge in the game domain and the model domain is presented. The challenges of game development, and the challenges of applying model driven techniques in game development, are briefly stated. The challenges found will be summarized, and research questions derived accordingly in Chapter 3.

Important concepts and development traditions of general computer games as well as emerging pervasive games are introduced. In addition, this chapter introduces elements, techniques, and the process of model driven software development. Also, a review is given of the present work of applying model techniques in game creation, for both game design and game software development.

2.1 Computer Game Concepts

This section introduces basic concepts for games, computer games, gameplay, as well as some conceptual frameworks of games.

2.1.1 Games

A game is firstly a type of play activity. Compared with other forms of entertainment, play is participatory while books, films, and theater are presentational [34]. However, a game, as a type of play, is strictly connected with rules (definitions and instructions that the players agree to accept for the duration of the game [34]). This means that, although a game includes the freedom to act and the freedom to choose how players act, this freedom is not unlimited. Instead, it is constrained by the rules, and this requires players to be clever, imaginative, or skillful in their play. As [34] defined, “Game is a type of play activity, conducted in the context of a pretended reality, in which the participant(s) try to achieve at least one arbitrary, nontrivial goal by acting in accordance with rules”. Similarly, Roger Caillois [35], inspired by Johan Huizinga [36], defined a game as “a fictional, unpredictable, and unproductive activity with rules, with time and space limits, and without obligation”. He also presented an approach for classifying games and especially identified two orientations which he called paida and ludus. We can understand them as freedom and constraints. Gonzalo Frasca [37] indicated that they described “the difference between play and game”.

Further, game is an interactive activity instead of a one-way activity. As defined by [38]: “A game is a voluntary interactive activity, in which one or more players follow rules that constrain their behaviour, enacting an artificial conflict that ends in a quantifiable outcome”. The interactions between a player and a (video)game can be understood as a continuous dialogue: “A cyclic process in which two active agents alternately (and metaphorically) listen, think, and speak” [39]. In [40], the author analysed many other definitions of game and summarized six important elements for a game as an interaction: 1) Rules: Games are rule-based; 2) Variable, quantifiable out-
Games have variable, quantifiable outcomes; 3) Value assigned to possible outcomes: That the different potential outcomes of the game are assigned different values, some being positive, some being negative; 4) Player effort: That the player invests effort in order to influence the outcome (i.e. games are challenging); 5) Player attached to outcome: That the players are attached to the outcomes of the game in the sense that a player will be the winner and "happy" if a positive outcome happens, and the loser and "unhappy" if a negative outcome happens; 6) Negotiable consequences: The game can be played with or without real-life consequences.

2.1.2 Computer Games

Juul’s definition of a game shows that games are trans-medial. “There is no single game medium, but rather a number of game media, each with its own strengths. The computer is simply the latest game medium to emerge” [40]. Games can be mediated by paper, board, and other physical mediums as well. [34] also defined a computer game from the medium perspective: “A video game is a game mediated by a computer” (many researchers use the terms computer game and video game interchangeably).

Using the computer as the medium brings a lot of benefits to games. Firstly, as games are “fictional” [35], “enacting an artificial conflict” [38], and presenting “a pretended reality” [34], computers can conveniently simulate such unreal artefacts. Therefore [37] defined video games as “simulations” and “a software system that models the behaviour of a real or fictional system”. Such a fictional system can also include a fictional story: “a videogame as a game which we play thanks to an audiovisual apparatus and which can be based on a story” [41]. Secondly, computers allow games to be more complex by automating many tasks and presenting a richer game world. As said by [42]: “the main difference between the computer game and its non-electronic precursors is that computer games add automation and complexity”, they can uphold and calculate game rules on their own (hiding the rules), allow for richer game worlds, set the pace, and create Artificial Intelligence (AI). Finally, the computer enables video games to borrow entertainment techniques from other media like books, film, karaoke, etc. [34]. As a result, computer games “are therefore part of the broader area of games, they have in many cases evolved beyond the classic game model” [40].

As the computer automates the rules, rules play an important role for computer games because “looking at games as rules means looking at games as formal systems, both in the sense that the rules are inner structures that constitute the games and also in the sense that the rules schemas are analytic tools that mathematically dissects games” [43]. This makes it possible to automate the software of such formal systems in the present research.

In the context of this thesis, “computer game” is preferred as this term is more precise since computer games can only enable audio presentation without visual presentations.

2.1.3 Gameplay and Mechanics

As mentioned in the previous section, a game is one type of interactive activity. Gameplay is probably the most important part of a game as it is “the degree and na-
ture of the interactivity that the game includes” [44], and it is “the component of the computer games that is found in no other art form: interactivity” [44]. Gameplay was considered to be implicitly defined by rules [43].

Besides gameplay, rules, as definitions and instructions that the players agree to accept for the duration of the game, can be used to define game semiotics, play sequence, game goals, termination conditions, and the meta-rules as well [34]. For video games, [37] there are three kinds of rules characterized: manipulation rules, goal rules and meta-rules. Manipulation rules are about how players interact with the game. Goal rules define how players can achieve game victory conditions. Meta-rules define how game rules would be modified, i.e., how players can change the game. Setting aside game meta-rules, which are not mandatory for gameplay definition, manipulation rules and goal rules are good intellectual tools for defining game system behaviour. Similarly, three levels of game rules were defined by [43]: operational rules, constitutive rules and implicit rules. Operational rules are the guidelines defining how players are required to play the game, i.e., the rules of play. Constitutive rules are about the underlying logical and mathematical structures of the game. And implicit rules are about the assumed etiquette conventions of good conduct among players. Compared with the three kinds of rules identified by [37], operational rules cover both manipulation rules and goal rules.

Corresponding to manipulation rules and goal rules, [34] also regarded play and goal as two of the essential elements and defined Gameplay as “consisting of challenges and actions that a game offers: challenges for the player to overcome and actions that let her overcome them”. Challenges were also considered as one kind of gameplay defined by rules in a game [45]. Similarly, gameplay was explained as “what players are allowed to do” and “how the game is played” by [44]. It is illustrated as game and play [46] which involve two kinds of videogame rules: rules “listening to Input and acting on the game elements consequently” named “Play bricks” and rules “observing the state of the game elements and returning to the player an evaluation of his performance” named “Game bricks”. These terminologies are used in similar contexts and semantics, as presented in Table 4. For the purpose of this research, [34]’s definition is chosen as it is the most formal and precise one.

<table>
<thead>
<tr>
<th>[34]</th>
<th>Gameplay</th>
<th>Challenges</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>[43]</td>
<td>operational rules</td>
<td>goal rules</td>
<td>manipulation rules</td>
</tr>
<tr>
<td>[37]</td>
<td></td>
<td>game bricks/ rules</td>
<td>play bricks/ rules</td>
</tr>
<tr>
<td>[46]</td>
<td>gameplay rules</td>
<td>challenges</td>
<td></td>
</tr>
<tr>
<td>[45]</td>
<td>gameplay</td>
<td>how the game is played</td>
<td>what players are allowed to do</td>
</tr>
</tbody>
</table>

Despite the importance of gameplay, the gameplay kernel in extremely fancy and complex computer games can be simple and straightforward. [1] gave several examples of gameplay for well-known games like Diablo and SuperMario Bros. In Diablo, “in an attempt to amass treasure and become more powerful, a player moves his character about an overhead map, battling any monsters or enemies it comes into contact with”. While in Super Mario Bros, “a player controls Mario (or Luigi), making him walk, run, and jump, while avoiding traps, overcoming obstacles, and gathering treas-
ure”. Further, while a game may consist of several gameplays [47], gameplay may be a part of many games (the authors used a similar terminology known as “mechanics”). The authors identified new mechanics and used them as tools to discuss game design, analysis and construction.

Following similar ideas, [34] identified several kinds of challenges such as physical coordination challenges, logical and mathematical challenges, race and time pressure challenges, factual knowledge challenges, memory challenges, pattern recognition challenges, exploration challenges (spatial awareness challenges, locked doors, traps, mazes and illogical spaces, tele-porters, finding hidden objects), conflict, economic challenges, conceptual reasoning and lateral thinking puzzles, etc. Similarly, [46] identified a set of gameplay recurrences within videogame rules named “Gameplay bricks” (play bricks and game bricks). The authors proposed to organize these bricks to build gameplay, which makes these bricks a good intellectual tool for defining game system behaviour (as demonstrated in Figure 1).
However, despite the fact that sometimes challenges can occur when performing actions in a pre-defined way, [34] suggested that the one-to-one mapping between actions and challenges should not always be expected: many games offer a large number of types of challenges but only a small number of actions, “leaving the player to figure out how to use the actions in various combinations to surmount each challenge”. The number of actions is often limited to avoid confusing players with a large user interface or having to pay for an expensive large number of animations to display.

Mechanics is sometimes used in the same way as gameplay, like in [44]. [47] also used Mechanics to describe game interaction, referring to “any part of the rule system of a game that covers one, and only one, possible kind of interaction that takes place during the game, be it general or specific”. But sometimes it is used for a wider scope or in a lower level, referring to particular components of the game such as various actions, behaviours and control mechanisms that are afforded to the player at the level of data representation and algorithms [48]. In [48], the authors proposed the MDA model (Figure 2), where mechanics together with the game’s content (levels, assets and so on) support overall gameplay dynamics (run-time behaviour of the mechanics), and then create game aesthetics (desirable emotional player responses). In [34], a gameplay mode was defined (see Figure 4) as consisting of “the particular subset of a game’s total gameplay that is available at any one time in the game, plus the user interface that presents that subset of the gameplay to the player”. As Figure 3 shows, mechanics (as lower level mechanisms) is used to implement the gameplay. Similar-
ly, in [49], a USE model was proposed (as Figure 3 shows) where mechanics was an essential element in the system (corresponding to the user and the game experience). In this thesis, [48]’s definition of mechanics is used to avoid ambiguity.

![Diagram of Gameplay Mode](image)

**Figure 4. Gameplay Mode [34]**

### 2.1.4 Vocabularies and Conceptual Frameworks

In an article published in 1999, Doug Church called for the development of ‘formal abstract design tools’ [16]. Since then, a number of frameworks and tools have sprung up. Many of them are primarily design vocabularies, which were created to help understand and identify common structures in games. These frameworks and tools tried to avoid being prescriptive in their description of games [50]. Other approaches include common mechanics, gameplay bricks (as introduced previously), and also design patterns applied from architecture and software engineering [51]. In contrast to vocabularies, *design patterns* are prescriptive as they describe good and generic solutions to common problems. However, the most prominent work [52] on this subject explicitly distanced itself from a prescriptive approach, creating a hybrid approach that was closer to a design vocabulary than a pattern language [50]. Some of the vocabularies found in this domain are introduced in this thesis, and include “taxonomy”, “dimensions”, “ontology”, “typology”, and “conceptual frameworks”.

In [53], the author described *orthogonal taxonomies* in order to separate game design concerns at the highest level compared to a story or a simulation. The authors proposed that orthogonal taxonomies were needed since “not everything falls into a simple hierarchical system of categories and subcategories. Orthogonal taxonomies allow design concerns to be separated”. Such orthogonal taxonomies included simulation, narratology, ludology, gambling, fiction and virtual/physical gaming (see Figure 5).

A feature model named the *NESI model* was presented in [4] and was based on four main features: Narrative, Entertainment, Simulation and Interaction. The NESI model was based on the game conceptual definitions proposed by [41], where a definition of videogame (combining existing research about game, play, interactivity, and narrative concepts) was described. NESI focused on the game domain, and was independent of graphics, audio, physics, artificial intelligence, or other type of game engine. The resulting hierarchy of features was quite comprehensive, and as the author stated, the great importance of this work has been to “reduce the gap between generic game domain descriptions and specific game project implementations”.

16
In [45], the Game Ontology Project (GOP) was described. GOP aimed to create a framework for describing, analysing and studying games. In GOP, a hierarchy of concepts abstracted from an analysis of many specific games was constructed and the top level of the ontology included concepts of interface, rules, goals, entities, and entity manipulation. Based on the concepts and terminology developed in GOP, the authors introduced in [54] the notion of gameplay segmentation (under the rules segmentation) to capture the role that design elements (such as level, boss, and wave play) have in videogames and to identify three modes of segmentation. In the same paper, temporal segmentation (which limits, synchronizes, and/or coordinates player activity over time), spatial segmentation (which breaks the game’s virtual space into sublocations), and challenge segmentation (which presents the player with a sequence of self-contained challenges) were also illustrated. In other research, further aspects were addressed like temporal frames [55] and spatial configurations [56]. The whole architecture and hundreds of definitions for all related concepts were described in the website [57].

[58] tried to establish a multi-dimensional typology to classify the genre of “games in virtual environments- that is, games that take place in some kind of simulated world, as opposed to purely abstract games like poker or blackjack”. The model consisted of basic dimensions like space, perspective, time, teleology, etc. Each of the dimensions had several variable values. The perspectives of this model were discussed in [59] with emphasis on how a structural theory of games could contribute to game design and the development of formal and semiformal game design methods, such as Game Design Patterns. The modified typology included eight meta-categories: virtual space, physical space, internal time, external time, player composition, player relation, struggle, and game state.

[60] also tried to construct a taxonomy for classifying games and simulations. Based on three interconnected building blocks (actors, rules, and resources), the author pointed out that even if games have similar forms, their purpose, subject matter, content, context of use, and intended audience(s) may be very different.
In [61], the author attempted an analytical and design understanding of contemporary play and games through the lens of architectural paradigms. By enumerating the various dimensions of game and play, and relating them to each other, the author built an *archaeology of ludic architectures*, and formulated some useful and meaningful questions. The framework is expected to work as a bridge between the disciplines of game design and architecture design.

### 2.2 Computer Game Development

In this section, the common process of game development is introduced, as well as challenges in this area.

![Figure 6. Model for Iterative Game Design: Playtest, Evaluate, and Revise [1]](image)

#### 2.2.1 Phases

Games are created in stages (some researchers use phases). [62] introduced four stages of game creation which include: the specification and planning (conception) stage, the pre-production stage, the development stage, and the validation and testing stage. The *conception* of a game starts with an original idea proposed by a studio or a query submitted by an editor. During this stage, a description about the main original game features, a draft of the planning and cost evaluation, as well as a first game prototype may be produced. At the end of this stage, a decision to carry on is taken mainly based on marketing considerations. The *pre-production* stage is mainly devoted to the game design and a significant prototype. The cost and planning may be refined and a second carry on decision is taken. Then the *development* stage includes creating all the game elements (images, sounds, video, and programs) and their integration.
(level design). The last stage (validation and testing) consists of functional testing (collectively known as alpha tests) which evaluates the quality of the game. Following the alpha tests, a debugging process (collectively known as beta tests) is performed by a specific team. [1]’s game creation model uses almost the same four stage names: concept, pre-production, production, and QA stages (see Figure 6).

In [34], the author used the same first stage (concept) and last stage (tuning, which corresponds to QA), but combined the in between pre-production and production/development stages into one elaboration stage (as Figure 7 indicates). The main reason for this combination is the highly iterative nature of both these two stages (indicated by the circle in Figure 7). Within these two stages, most of the design details are added and decisions are refined through prototyping and play-testing (this will be discussed further in the following section). Since game design, level design and software development are tightly connected and are enhanced by each other, it is difficult to find a specific time separating the pre-production stage from the production stage.

![Figure 7. Three Stages of the Game Design Process [34]](image)

[34] also enumerated the main tasks for each stage. During the concept stage, the main tasks are acquiring a concept, defining an audience, determining the player’s role, and fulfilling the dream. While in the elaboration stage, tasks would include: defining the primary gameplay mode (as mentioned previously), defining the protagonist, defining the game world, designing the core mechanics, creating additional gameplay models, designing the levels, writing the story, etc. These tasks are synchronized with the descriptions of [62] as well, where the last stage is called a tuning stage (since at that point no new features would be added, and only small adjustments can be made to polish the game).

The research of this thesis focuses on the elaboration stage because it does not involve much planning, marketing, ideas, or initiating work, but counts for most of the concrete designing and development work. Although it is difficult to isolate the pre-production and production stages from the temporal perspective, they still are distinguished in this thesis as their different foci (game design and level design) play important roles in the overall game creation process. In this thesis, [1]’s stage names are therefore used: concept stage, pre-production stage, production stage and QA stage.

The stages mentioned above are also in accordance with the Rational Unified Process [63] stages for the common project life cycle phases: the inception phase, elaboration phase, construction phase, and transition phase (Table 5 summarizes the game creation stages). Looking at game creation as a collection of common software projects helps us to understand game creation better by borrowing mature project development methodologies and experiences. Rational Unified Process (RUP) is a software process product, originally developed by Rational Software, which was acquired by IBM in
February 2003. In RUP, there are three kinds of building blocks [63]: roles (who), work products (what), and tasks (how). In game creation, multiple roles from different disciplines work together (in different ways) to create a game (consisting of various work products). In this thesis, the focus is on those building blocks relating to the design and software parts (not the marketing, artistic, planning, and testing parts). To be specific, the emphasis is on the roles of the game designer, level designer and programmers. Of particular interest are the work products of game design documents and software programs, and the tasks of game design, software design and software development.

<table>
<thead>
<tr>
<th>Table 5. Game Design Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUP</td>
</tr>
<tr>
<td>Inception phase</td>
</tr>
<tr>
<td>Elaboration phase</td>
</tr>
<tr>
<td>Construction phase</td>
</tr>
<tr>
<td>Transition phase</td>
</tr>
</tbody>
</table>

2.2.2 Iterations

Games are created for entertainment and the most important requirement of a game is to entertain players. This makes a game differ from common productivity applications in many ways. In [64], the authors discussed in detail the contrast between video games and applications. [65] summarized the interaction-centric points, and stated that the degree of innovation of games tends to exceed that of productivity applications. The sounds and graphics in games are meant to convey moods instead of functionality. Games impose constraints on users instead of removing them. The goals of games are usually defined and motivated within the game world instead of outside the application. Most importantly, games focus on the process of use (gameplay) rather than the result. Further, games actively encourage various (not consistent) experiences.

“The process of use” and “various experiences” are not easily defined before a system can really be used. This may lead to gameplay being the main reason as to why the game process is performed in iterations. During the iterations, prototypes are evaluated and enhanced [62] for both the design and the programs. Similar opinions presented in the literature state that “it is always difficult to say whether a gameplay is really playable or not unless you can really play it [34]” and “the more inexperienced the game designers are, the more difficult it is to imagine what kind of gameplay will emerge when the players, and potentially the environment, interacts with it” [66]. Also, games must be designed with a gradually increased degree of difficulty. Hence, as a consequence, games “are generally designed and developed in an increasing order of complexity” [62].

In addition, since game projects are facing evaluation all the time and may be aborted at any time [62], agile iterations provide an efficient way to shorten the time to decision and lower the cost at any time. They help to demonstrate, test, generate game
design ideas, and probe the attitudes of potential players [66]. Therefore, (iterations and) prototyping is a commonly used design method in game development [1].

2.2.3 Game Design and Level Design

As mentioned previously, during the pre-production and production phases, the main tasks are game design, game elements creation and integration (level design), and iterative evaluation and enhancement. As introduced by [62], a game is an imaginary universe. Game design defines the main aspect of this universe: game context (epoch, style, historical or mythical references), global scenario (typology, main characters), main features, gameplay principles, goals, object classes (object oriented approach), etc.

While game design defines the gameplay (goals and actions) principles and classes of the game world objects (object oriented), level design chooses the main actions and goals for the player, and mixes them with the virtual space and concrete world objects [62]. Despite the notion of “level”, the relationship between level and level design may not be as straightforward as a one-one relationship (level design does not always generate levels). The main task of level design is tuning architecture and gameplay (as will be introduced below), and this task is always needed, even for games that do not have more than one level. From this point of view, we can understand level design as a specific part of game design – and for complex games, especially those with many levels, this part of design and the design results can be easily separated (to levels). On the other hand, even for games with more than one level, not all of these levels have to be pre-defined (through level design during game creation phases). Some games generate levels automatically when they are played using internal rules to make sure generated levels are interesting [44]. Also, some games (especially build games) leave large parts of level building work to the players [44].

The pre-defined levels, are usually developed in the level editor that is provided by a game engine package [44]. [44] introduced the procedure to develop a level, as Figure 8 shows. In the procedure, we can see that the main tasks are to refine architecture and gameplay (the two circles) until they are fun. [62] mentioned that level design approaches are mainly scenario oriented in the present game industry. The notion of scenario comes from the movie world and is related to a story and a (time driven) sequence of scenes. The importance of scenario may vary for different genres of the game. For example, a scenario often drives adventure games, but in action or simulation games the scenario may only be a part of the context. However, there is always a scenario in any game even if it may be reduced to a sequence of goals, or a circuit the player must follow during a race in a car game [62].

![Figure 8. Level Design Process](image-url)
[34] also used scenario to design levels. Furthermore, the author defined gameplay mode as the combination of the scenario and interaction elements: “a gameplay mode consists of the particular subset of a game’s total gameplay that is available at any one time in the game, plus the user interface that presents that subset of the gameplay to the player” (see Figure 9). The author considered gameplay modes as being central to the process of designing video games. The typical way to use gameplay modes to design a game was also illustrated as: “to define the actions, firstly you need to think about the player’s role and what is the player going to do, and look at the challenges in the primary gameplay mode…once you have been through this process for the primary gameplay mode, do it all again for each of the other modes…then you can start defining the user interfaces for different modes: assigning actions to control mechanisms.” As the author did not distinguish game design and level design explicitly, it can be understood that this gameplay mode oriented design approach is practically available for both of them.

![Figure 9. Gameplay Mode [34]](image)

### 2.2.4 Prototypes

Prototypes are used commonly as a design method in game development. Prototypes do not have to be realized by software. [1] identified two kinds of prototypes: physical prototypes and software prototypes. The authors stated that “software prototypes are analogous to physical prototypes, except that they are made using programming tools”. [66] presented three kinds of prototypes, including physical prototypes, prototypes with rapid software development, and prototypes using ready-made software. [1] also indicated that experimenting with level editors (as one kind of ready-made software which comes with games or is created by third parties) is a useful and interesting way to learn about prototyping games. The authors also presented guidelines as to how to select the right prototyping method based on the purpose of the prototype, the game type (as Table 6 presents), the project type, etc.
Table 6. Selecting Prototyping Methods Based on Game Type [66]

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

Physical prototypes are cheap but do not always help us to really get a feel for a game. [1] used a simple example of the Tetris game. Although Tetris originates from a physical puzzle, it is tightly bound to interactions with the computer which are difficult to model elsewhere. In this case, a physical or paper prototype would be more difficult to construct than a software prototype. Further, these prototypes are primarily used in the pre-production phase in order to test and improve kernel concepts. Such prototypes cannot be used in the final game product and they cannot be used to tune gameplay etc. during the process of developing levels. As stated by [1], (software) prototypes allow us to visualize how gameplay actually works in ways that we would not be able to see with just paper prototypes.

In the research of this thesis, the focus is on software prototypes. That is to say, the prototypes that are used to validate or tune game/level design. The software of such prototypes is not discarded. Instead, the software grows together with the design until the game product is finished. Among software prototypes, the baseline prototype is defined as the first working version of the formal system [1] that is created (for validating main game design and programming techniques) during the pre-production phase. Additionally, tuning prototypes are defined as the other prototypes used in the production phase (for tuning gameplay in levels). How to enhance the development of such prototypes is one of the research focuses of this thesis.

Baseline prototype defines the core mechanic and allows us to test it through play. It “lies at the heart of good game design” [1], “teaches you about the essence of game
design”, “helps you understand how the activity becomes meaningful over time”, and thus it should define the core mechanics in their purest form with the only consideration being to get a game to work. As a result, the baseline prototype should be able to keep you engaged (as a crude model with the essence of gameplay and having unpolished art) and be functional (allowing to test through play). Indeed, for most of the great games, the core gameplay is not complex.

Tuning prototypes are the main tools used for play-testing, and the goal of play-testing is to gain useful feedback from players in order to improve your game. [1] defined play-testing as “something the designer performs throughout the entire design process to gain an insight into how players experience the game”. In addition, the authors mentioned that there are numerous ways to conduct play-testing, such as using an informal or formal structure, and using qualitative or quantitative methods. Such tuning prototypes are part of the final products (if the game is not aborted). In these prototypes, architecture, gameplay, and some of the art work are polished. But the stories are not. As said by [44]: “we prototype gameplay, but not story although both are expected to be non-linear and should match each other as much as possible, but story is easier to understand and communicate”.

2.2.5 Software Architecture

Due to the game domain diversity, researchers have tried to manage the game development variability by many means. The most successful approach in the industry was first achieved by game engines [4]. A game engine separates game content (like art, music and character behaviour) from more common and lower level mechanisms (like multimedia manipulation and event handling) and often is located at the heart of the overall software architecture [12]. [62] depicted a typical architecture for such commercial video games, as Figure 10 shows. In such a game, there are three levels of programming: script code, gameplay code, and engine code. Script and gameplay code control the overall content and high-level behaviour of the game. This part of the code is often authorized in a level editor, which is available within the game engine package. Therefore, game assets are portable between games, but the games are dependent on the game engine they were developed on.

An engine can be seen as a reusable Application Program Interface (API) for common game development building blocks [67]. These building blocks include logic for graphics, sounds, and physics. In addition, this logic is usually built on top of traditional multimedia APIs to free developers from low-level implementation details. In order to be more effective, most game engines narrow their target domain to a specific subset of all possible computer game genres [68]. For example, a 3D game engine has many different issues from those of a 2D game engine. [69] also presented similar introductions to game engines. Popular game engines include OGRE [70] and Crystal Space [71].

However, as [72] pointed out, game engines have some drawbacks as well. First, the lack of usability may be one of the most cited deficiencies of game engines. As game engines are inherently complex to satisfy a variety of requirements, it is not intuitive and easy to understand the game engine architecture, interaction paradigm, and programming peculiarities. Using a game engine involves considerable efforts towards acquisition, training, customization and integration. And the learning curve for mastering these tasks is usually quite high. Second, as a game engine usually targets a
specific game genre, using a game engine often involves writing/modifying game engine codes to fit the needs for developing a specific game. The effort used to write/modify a game engine can be even larger than for a specific game due to the complexity of game engines. Finally, game engines are often industrial secrets. Many companies and organizations hide their architectures and tools to have advantages over their competitors [73]. For example, it is difficult to find comprehensive studies about the applicability of design patterns in game engines [68]. Public knowledge is only available through open source and academic initiatives. Developers are far from having something like “engine workbenches” to aid their game engine creation.

<table>
<thead>
<tr>
<th>Architecture Levels</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level design scripts editors</td>
<td>God move(right, 2).Wait Event; On button.click (God_Anger==new(thunder)) God_Anger:lightning, God_Anger:source</td>
</tr>
<tr>
<td>Game classes</td>
<td>class thunder methods: lightning, sound</td>
</tr>
<tr>
<td>Game engine library: general purposes game oriented libraries</td>
<td>Create_new_object(God, god_geometry.vrml, god_texture.gif, god_voice.wav)</td>
</tr>
<tr>
<td>Graphic engine</td>
<td>(DirectX, OpenGL, Open AL, ...)</td>
</tr>
<tr>
<td>Sound engine</td>
<td></td>
</tr>
<tr>
<td>physic Engine</td>
<td></td>
</tr>
<tr>
<td>IA engine</td>
<td></td>
</tr>
<tr>
<td>General purpose multimedia libraries</td>
<td>GLMatrixModel(); alsourceplay(source1)</td>
</tr>
<tr>
<td>Operating system</td>
<td>Windows, PS2 Monitor...</td>
</tr>
<tr>
<td>Central Hardware Processor, memory...</td>
<td></td>
</tr>
<tr>
<td>Graphic accelerator</td>
<td></td>
</tr>
<tr>
<td>Sound card</td>
<td>PC, PS, GameCube, XBox</td>
</tr>
</tbody>
</table>

Figure 10. Game Software Architecture [62]

Multimedia APIs play another important part in game software architecture. These programming libraries provide standard means to access the machine hardware directly (graphics devices, sound cards, etc.). With this standard device manipulation interface, programmers do not need to worry about the low-level peculiarities of each possible device. Therefore multimedia APIs help to improve game performance and enable portability among devices from different vendors. Well-known examples are Microsoft DirectX [74] and OpenGL [75].

Multimedia APIs can be used directly. However, as these libraries focus on very-low level functions, they do not offer the proper abstraction level desired by game programmers. The gap between game designers and the final code remains quite high. In addition, interaction with such APIs can not be done in a visualized way. This may bring difficulties to the automation and productivity improvement of some work. Some work, like specifying the tiles of a background map, would have to be executed by “exhaustive copy and paste commands and through counter-intuitive actions” [76]. Therefore, multimedia APIs are more often used indirectly (through game engine).

2.2.6 Game Authoring Tools

To simplify game development and enable more people without an intensive programming background, like game designers, to create games by themselves, various
game authoring tools have emerged. Usually these tools provide a visualized interface to enable developers to create a game with simple operations like a few mouse clicks.

However, as indicated by [67], these tools do not address the complexity required by more sophisticated games, and thus are not adopted by the game industry widely. They are very popular, but the main users are beginner and amateur game designers. Some of these tools try to address the complexity required by sophisticated games, and offer some kind of scripting languages. But this requires end-users to learn a new language. As a result, basic users may not use such scripting languages, while advanced users will prefer to use some true object-oriented programming languages with generic and better IDE support. In addition, providing scripting languages diverges from the original purpose of the authoring tools (to be visual and user friendly).

However, some of these tools are deliberately designed and implemented within a narrower domain, aimed at some specific game genre, focused on specific game aspects, providing simple operations, targeting specific audiences, etc. Not surprisingly, due to the limitation of generality and difficulty of customization, finding fitting tools may not be easy. You may not always find a proper tool for your specific game domain, or for some specific requirements. In addition, you still have to use time to learn and use unnecessary features that the tools sometimes provide. In the following section, several representative authoring tools are introduced.

**Figure 11. PlayMaker [77]**

*Playmaker* [78] is a powerful visual state machine editor and runtime library for Unity3D. The tool provides the means to specify AI behaviours, animation graphs, interactive objects, in-engine cut-scenes, gameplay prototypes, interactive walkthroughs, etc. It is possible to create games without knowing programming, but it also provides scripts for advanced users. Figure 11 shows what the typical UI looks like.

Similarly, *GameMaker: Studio* [79] provides both easy interface (see Figure 12) and build-in scripting language (Game Maker Language (GML)) in order to meet requirements from people with different programming backgrounds. The tool allows users to create cross-platform games quickly. Another example of such authoring
tools is the *RPG Maker* series [80]. This tool allows beginners and experts alike to customize various aspects of the game with an easy to use interface (see Figure 13).

![Figure 12. The GameMaker: Studio Environment [79]](image)

![Figure 13. RPG Maker XP ENTERBRAIN 2013 [80]](image)

*Inform* [81] is a textual based IDE (see Figure 14) for interactive fiction creation. “It is a radical reinvention of the way interactive fiction is designed, guided by contemporary work in semantics and by the practical experience of some of the world's best-known writers of IF” [81]. *Articy:draft* [82] claims that it is the first professional tool (see Figure 15) for story and game design which provides the means to create non-linear story design and dialogues design, character and object design, as well as worlds, locations and sets design. Similarly, *Chat Mapper* [83] focuses on writing and testing nonlinear dialogue and events, especially for video games and training (see Figure 16).
Figure 14. Inform7 [81]

Figure 15. Articy:draft - Interactive Stories and Dialogues [82]

Figure 16. Chat Mapper– Non-Linear Branching Tree Graph Visualization [83]
Dash [84] is a visual programming tool (see Figure 17) and a game creator. It is based on HTML5 and can run on any device. Dash is well suited for simple and single player 2D arcade games, and supports features like the dynamic creation of objects, 2D physics, particles, scene editing, and animation editing. Kodu [85] lets kids create games on the PC and Xbox with no design or programming skills required. It can be used to teach creativity, problem solving, and storytelling, as well as programming. Figure 18 presents what Kodu’s main user interface looks like. Scratch [86] is aimed at beginners of game creation, and young people as well. It can be used to create interactive stories, animations, games, music, and art, as well as to share creations on the web. By creating and sharing Scratch projects, young people may learn important mathematical and computational ideas, and to think creatively, reason systematically, and work collaboratively. All operations within the IDE (see Figure 19) are extremely intuitive, and the applications are ready to run once you finish building the blocks; you can even modify them while the application is running.
2.2.7 Challenges

Game development is becoming increasingly difficult as more commercial computer games have grown to be larger and more complex in recent years. As said in [5], “Ten or twenty years ago it was all fun and games. Now it’s blood, sweat, and code”. [5] identified two main reasons for game development difficulties: the overall big project size and high complexity, and the highly domain-specific requirements. The author raised problems caused by the overall project size and complexity, such as:

- **Tools.** We do not have ideal development tools to tackle the complexity.
- **Workflow.** We have many workflow problems.
- **Multiplatform Development.** Many games are developed to run on multiple platforms. This makes the development even more complex.
- **Third-Party Components.** It is not easy to reduce the workflow to leverage third-party products.
- **Full-Figure Option.** Instead of licensing components, we can license an entire game engine from a company that has successfully built a solid one.

The problems that occur due to highly domain specific requirements include:

- **Engine Code.** Writing good engine code requires a good grasp of software engineering and a lot of domain-specific knowledge. This is not easy.
- **Crosscutting Concerns.** When many algorithms are put together into a tightly coupled system, constraints imposed by the various algorithms may clash.
- **Depth of Simulation.** We need to cover the simulation of more aspects than before, such as physics and AI, which can be more important than the graphics to the players’ experience.
- **Profiling.** We do not have a mature profiling tool to help us use the CPU to the full extent possible. Usually, we build our own simple profiling systems into our games.
- **Risk.** To try to give players gameplays that they have never experienced before, each game attempts technical feats that are mysterious and unproven. Game developers carry a lot of technical risk (to schedule the unknown tasks or predict an interaction) and game design risk (how the end user will feel or
will it be worth all the effort in the end).

Researchers have tried to highlight common issues and challenges which exist in current game development. [88] outlined three main issues caused by component based architecture: loss of information about the domain, entity inconsistencies at a component level, and inconsistencies due to attribute definitions. [87], however, focused more on the process aspect. The authors pointed out that the software engineering process has not been clearly understood in video game development, which hinders the development of reliable practices and processes for this field. In the paper, the game development process was modelled as two consecutive efforts (see Figure 20). The left hand side of the diagram depicts the preproduction phase, which results in a Game Design Document (GDD). This preproduction phase loosely corresponds to customers’ internal efforts to define their wants and needs before they meet with the development team. The right hand side of the diagram depicts the production phase, when GDD is transformed to a specification by requirements engineering with the assistance of game designers. Once the specification is complete, a traditional software development process begins. However, moving from preproduction to production is difficult in video game development. The authors highlighted three specific problems and concluded that traditional requirements engineering techniques need to be extended to support the creative process in video game development. The three problems are:

1) How to transform documentation from its preproduction form to a form that can be used as a basis for production;
2) How to identify implied information in preproduction documents; and
3) How to apply domain knowledge without hindering the creative process.

These issues are also indicated in other scientific papers. In [89], the authors indicated that many game project reports showed that game production is a complex task and is distant from having a healthy and synergetic work process. The authors collected problems related to the development process of computer games (“electronic games” in the article), and explored their similarities and differences with well-known problems in traditional information systems. From the article, problems in the software industry were categorized into four main groups: scheduling problems, budget problems, quality problems, and management and business related problems. In addition, fifteen main problems were summarized and ordered by analysing game project re-
ports (see Figure 21). As can be seen in Figure 21, the game problems that occurred the most were unrealistic scope, feature creeping and feature cutting.

![Figure 21. Occurrence of Game Project Problems [89]](image)

By comparing the traditional software industry domain and the game industry domain, the authors indicated that all of the main problems found in the traditional software industry were also found in the game industry. Unrealistic scope was highlighted in both, as well as requirements analysis problems. When discussing differences between the two industries, communication among teams was one important problem specific to the game industry. In contrast to the relatively homogeneous team setup in the traditional software engineering industry, most game teams are multidisciplinary with “artists” and “programmers”. This can cause important communication problems since the two teams may use different vocabularies to express their ideas, and different vocabularies are an important source of misunderstanding. In addition, elaborating game requirements is much more complex because there is no efficient method to determine subjective elements like “fun”.

### 2.3 Pervasive Game Concepts and Development

In this section, the emerging game genre is introduced: pervasive games. In addition, some state-of-the-art design and development research in this area is presented.

#### 2.3.1 Origins, Elements and Definitions

The word pervasive has a clear meaning in the dictionary. It is an adjective meaning (the quality of) any given object or concept to spread, diffuse, or go through an area or a group of people [20]. The concept of pervasive game has emerged during the past decade. Researchers have used this concept to indicate that (computer mediated) game elements are spreading, diffusing, or going through our everyday life.
In [21], the authors indicated that before computer games appeared, (physical) games were designed and played out in the physical world, relying on real world properties such as physical objects, physical space, etc. However, computer games became a dominating form of entertainment as they provided a higher level of attractiveness. First, they create an imaginative virtual world with graphics and sound which allows people to become immersed in the illusion. Second, they provide more interactive goals to cause players to have a stronger desire to win. Third, they provoke players’ curiosity more easily with optimal information complexity.

On the other hand, computer games have often decreased the users’ physical activities and social interactions. This is mainly due to the usage of computers screens, mice and keyboards. The growing trend of pervasive games to bring more physical movement and social interaction into games is overcoming these shortcomings, while still utilizing the benefits of computing and graphical systems. As a result, pervasive games can be thought of as real-world games augmented with computing functionality, or depending on the perspective, purely computer games brought back to the real world [21].

In [90], the author described and analysed the formalisms of pervasive games and pervasive gaming (PG). The author indicated that PG consists of atomic entities. The article introduced four axes of PG which include mobility, distribution, persistence and transmediality (see Figure 22). Further, it described three key units of PG: rules, entities and mechanics (see Figure 23). The role of space in PG is discussed as well by differencing between tangible space, information-embedded space and accessibility space.

In addition, the author tried to propose a general or classic definition of PG: “Pervasive gaming implies the construction and enactment of augmented and/or embedded game worlds that reside on the threshold between tangible and immaterial space, which may further include adatronics, embedded software, and information systems in order to facilitate a "natural" environment for game-play that ensures the explicitness of computational procedures in a postscreen setting” [90].
However, some research has not taken it for granted that pervasive games should be primarily computer games. Instead, efforts have been made to find an expansion of such games over traditional physical games without emphasizing the usage of computers or computations. In [91], the author gave the definition of pervasive games from the perspective of the magic circle:

Pervasive game is a game that has one or more salient features that expand the contractual magic circle of play socially, spatially or temporally [91].

The metaphorical concept of the magic circle of play means a voluntary, contractual structure that is limited in time and space. Here the term of “the magic circle, or in a self-explanatory word, pretended reality, refers to the boundary that divides ideas and activities that are meaningful in the game from those that are meaningful in the real world. The magic circle defines the boundary between reality and make-believe” [92]. The author has indicated that breaking these boundaries of game is not an original idea, but that systematically breaking the boundaries makes pervasive gaming a novel gaming form. Similarly, [93] and [94] provided some insights on pervasive games from the magic circle perspective as well.

Actually, there are numerous definitions for pervasive games. As concluded by [20], pervasive gaming is understood in the literature as being:

- “a game that depends primarily on pervasive technology and nonstandard input devices;
- an existing game that is augmented by computers, resulting in a blend of the real and virtual worlds;
- a game that pervades the real world in an undefined manner, and thus blends with it;
- a specific setting of the game world within the real world;
- a game that blurs the boundaries between itself and the real world;
- a game that is an overlay of the real world;
• a game with a persistent presence in the real world, and thus available to the
  players at all times;
• a game where the gameplay interacts with elements of the real world, thus
  challenging standard gameplay conventions;
• a game where there is mutual interaction among players and elements in the
  real world;
• a game that blends with everyday experiences”.

These definitions do not refer to the same kind of games. In fact, some of the defini-
tions conflict with one another. This may explain why [20] critically analysed and re-
evaluated the term pervasive, from both the perspectives of computing and gaming.
The result showed that from both perspectives, ambiguous definitions exist, and the
term has become theoretically entangled with others. Further, multiple definitions ne-
glect the different perspectives in which a game can be pervasive. As a result, the au-
thor suggested that we should be aware of the context in which the term pervasive
was, and continues to be, used. Further, the author argued that “we need to let go of
the notion of coining the ultimate definition of pervasive games, and instead ask our-
ourselves what makes a game pervasive”. This gave rise to the concept of pervasiveness,
which is seen as “a characteristic that occurs in different perspectives and levels of
intensity and can be applied to different genres, games, and play”.

2.3.2 Genres and Cases

In [90], the author considered the following post screen gaming subgenres when de-
fining pervasive as an overarching concept or activity:
• Mobile games
• Location-based games
• Ubiquitous games
• Virtual reality games
• Augmented reality games and mixed reality games
• Adaptronic games

Similarly, [21] looked at five main sub-genres of pervasive gaming, including:
• Smart toys
• Affective games
• Tabletop games
• Location-aware games
• Augmented reality games

The authors discussed these games in terms of benefits, critical issues, and the rele-
vant technology bases. [95] also enumerated some subgenres of pervasive games,
mainly from the game design perspective.

Famous pervasive games include: The Drop [96], Capture the Flag [97] (a mixed real-
ity social game with smart phones), SupaFly [98], Epidemic menace [99] (a cross me-
dia game), Mobio threat [100] and Neat-o-Games [101]. Most of these are location-
based mobile games.
Table 7. Summary of Location-based Games [102]

<table>
<thead>
<tr>
<th>Game</th>
<th>Location technology</th>
<th>Location (May 2005)</th>
<th>Platform</th>
<th>Single &amp;/or Multiplayer</th>
<th>Available from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Botfighters 1 &amp; Botfighters 2</td>
<td>Cell ID</td>
<td>Sweden, Russia, China</td>
<td>SMS / J2ME</td>
<td>S &amp; M</td>
<td>2000 &amp; 2005</td>
</tr>
<tr>
<td>Gunslingers 1 &amp; Gunslingers 2</td>
<td>Cell-ID &amp; GPS</td>
<td>Singapore</td>
<td>SMS/ J2ME</td>
<td>S &amp; M</td>
<td>2003</td>
</tr>
<tr>
<td>Mogi</td>
<td>GPS</td>
<td>Japan</td>
<td>J2ME</td>
<td>S &amp; M</td>
<td>2003</td>
</tr>
<tr>
<td>Undercover 1 &amp; Undercover 2</td>
<td>Cell ID</td>
<td>Hong Kong Portugal</td>
<td>J2ME,</td>
<td>S &amp; M</td>
<td>2003</td>
</tr>
<tr>
<td>Swordfish</td>
<td>A-GPS</td>
<td>Canada/ USA</td>
<td>J2ME</td>
<td>S</td>
<td>2004/ 2005</td>
</tr>
<tr>
<td>Torpedo Bay</td>
<td>A-GPS</td>
<td>Canada/ USA</td>
<td>J2ME</td>
<td>S &amp; M</td>
<td>2005</td>
</tr>
<tr>
<td>Conqwest</td>
<td>Semacodes</td>
<td>USA</td>
<td>WAP</td>
<td>M</td>
<td>2004</td>
</tr>
<tr>
<td>The Journey</td>
<td>Cell ID</td>
<td>Austria</td>
<td>Symbian OS / UIQ</td>
<td>S</td>
<td>2004</td>
</tr>
<tr>
<td>Songs of the North</td>
<td>Cell ID</td>
<td>Trials in Finland</td>
<td>J2ME</td>
<td>M</td>
<td>Concept only!</td>
</tr>
<tr>
<td>Treasure Hunt</td>
<td>GPS</td>
<td>USA</td>
<td></td>
<td></td>
<td>2004</td>
</tr>
<tr>
<td>I Like Frank</td>
<td>Cell ID*</td>
<td>Adelaide</td>
<td>J2ME</td>
<td>M</td>
<td>2004</td>
</tr>
<tr>
<td>RayGun</td>
<td>GPS</td>
<td>USA</td>
<td>J2ME</td>
<td>S</td>
<td>2005</td>
</tr>
<tr>
<td>Tron</td>
<td>GPS</td>
<td>Trials in Austria</td>
<td>J2ME</td>
<td>M</td>
<td>Expected Q4 2005</td>
</tr>
<tr>
<td>Frequency 1550</td>
<td>GPS</td>
<td>Amsterdam</td>
<td>J2ME*</td>
<td>M</td>
<td>2005</td>
</tr>
<tr>
<td>Colors</td>
<td>GPS</td>
<td>Windows CE</td>
<td></td>
<td>S &amp; M</td>
<td>2004</td>
</tr>
</tbody>
</table>

Location-based games, as an important subgenre of pervasive games, were introduced in detail in [102]. The authors indicated that such games should address specific game development concerns such as:

- “within games, movement is mostly designed as a necessity;
- games cannot simply be consumed; to entertain, they constantly need the players to act;
- games often require other users to function;
- location-based games are not able to introduce a story;
- the ability of location-based games to handle player network latency in fast-moving games” [102].

In addition, technical issues were discussed, and many classic location-based games were analysed (see Table 7).
2.3.3 Design and Development

Since most researchers have regarded pervasive games to be a special genre of computer games, the majority of design and development is similar to that of computer games. However, a few researchers have presented initial trials to address the special attributes of pervasive games. For instance, in [21], the authors discussed some technology-based and critical issues according to the different genres of pervasive games. Similarly, [102] analysed technology issues according to location-based sub genres of pervasive games. For context awareness in particular, the authors in [103] introduced a structuring and presentation framework for use in context-aware mixed reality applications. The framework was based on a generic hypermedia model which handled different media elements, objects, and relations between spaces and locations within physical and virtual worlds.

![Figure 24. An Example of Gameflow Design [104]](image)

Besides these technical details, as player enjoyment may be the most important issue in successful game design, [105] outlined a model of pervasive player enjoyment. This model was based on the general gameflow model [49, 106], which was widely used and derived from a general flow theory raised by [107]. In [107], flow was defined as “an experience so gratifying that people are willing to do it for its own sake, with little concern for what they will get out of it, even when it is difficult or dangerous”. Flow experiences consist of eight elements, as follows:

1. a task that can be completed;
2. the ability to concentrate on the task;
3. that concentration is possible because the task has clear goals;
4. that concentration is possible because the task provides immediate feedback;
5. the ability to exercise a sense of control over actions;
6. a deep but effortless involvement that removes awareness of the frustrations of everyday life;
7. concern for self disappears, but sense of self emerges stronger afterwards; and
8. the sense of the duration of time is altered [107].

In [104], a formal and visualized approach to design computer gameflow (with not exactly the same meaning as the gameflow in [49, 106]) was introduced. Figure 24 shows an example of such a diagram.
Considering the specific requirements of adopting different prototyping methods for pervasive games, [66] discussed various prototyping methods and gave guidelines to help the selection process, as well as ideas on methods that can be used in different situations. The main methods include rapid game development, prototyping with ready-made software, and (guided) physical prototyping. The selection guidelines are based on various variants like desired result, prototyping requirements, game type, project-related issues, etc.

2.4 Model and Model Driven Development

In this section, background knowledge regarding model driven development is introduced, including basic concepts, elements and the process of model driven development, etc.

2.4.1 Model Preliminaries

In this section, basic concepts for model, modelling, conceptual modelling, model driven software development, and domain specific modelling are clarified.

2.4.1.1 Model, Modelling, and Conceptual Modelling

A model can be almost anything. Usually it is more abstract, less complex and cheaper to make than what it models. In [108], a model is understood as an abstraction of some aspect of a system. The system may or may not exist when the model is created. Thus models can be created for different purposes, for example, to present a human understandable description of a system aspect, or to present information in a machine analysable form for further processing.

Conceptual models are often used for representation and creation of knowledge. Such models are primarily diagrammatical as they can be more expressive. As said by [109]: “One can argue that the main reason why humans have excelled as species is our ability to represent, reuse and transfer knowledge across time and space. Whereas in most areas of human conduct, one-dimensional natural language is used to express and share knowledge, we see the need for and use of two and many-dimensional representational forms to be on the rise. One such representational form is called conceptual modelling”. A conceptual model is “a description of the phenomena in a domain” at some level of abstraction. And it is usually expressed in a semiformal or formal diagrammatical language. The author enumerated eight perspectives (the main phenomena models may describe) of models: behavioural perspective, functional perspective, structural perspective, goal and rule perspective, object perspective, communication perspective, actor and role perspective, and topological perspective.

Consequently, conceptual models can be used in various usage areas like [109]:

- Human sense-making;
- Communication between people in one organization;
- Computer-assisted analysis;
- Quality assurance, to ensure that an organization acts according to a certified process through an ISO-certification process model (for instance).
- Model deployment and activation. Models can be activated in three ways:
- Through people guided by process 'maps' where the system offers no active support.
- Automatically, where the system plays an active role (as in most automated workflow systems).
- Interactively, where the system and the people co-operate (the computer makes decisions while the users resolve ambiguities).

- Provide context for traditional system development project without being activated.

As mentioned above, models are not only useful for knowledge representation, but can also play more active roles, including designing software or even transforming into software. As said in [110], “a model plays the analogous role in software development that blueprints and other plans (site maps, elevations, physical models) plays in the building of a skyscraper”. And a model is understood as the design of software application before the coding. Adopting models for software design may mainly occur because models allow us to work at a higher level of abstraction by hiding or masking details, bringing out the big picture, or focusing on different aspects of the prototype [110]. For instance, in Universal Modelling language (UML) 2.0, it is possible to zoom out from a detailed view to the big picture, in order to visualize connections to other applications. It is also possible to focus on various aspects of the application alternatively.

Typical models include problem diagrams for problem analysis, feature model diagrams for requirement definitions (and possible further usage), and domain models for structural design (and possible further usage). These concepts are further elaborated on as they relate to the research of this thesis. The concept of the problem frame was proposed by [111]. Physical domains and phenomena are emphasized as the author assumed that software is utilized to build machines which interact with the physical world and change it. Thus the software requirement is expected to be fulfilled by observable physical world effects which are far from the machine.

An example of a problem diagram is presented in Figure 25. In a problem diagram, some concepts are defined. There may be three kinds of domains (which are represented by rectangles): the machine domain is the computer (represented by a rectangle with a double vertical stripe), the design domain is the physical representation of some information that we are designing (represented by a rectangle with a vertical stripe), and the given domain is the existing domains that you are not free to design (represented by a common rectangle). Between domains, there are interfaces, mean-
ing shared phenomena which include a set of shared events, states and values. These interfaces are represented as lines between the domains. At last, dashed ovals represent requirements (represented by dashed arrows connecting them to domains known as requirements references). A requirement refers to some phenomena of a domain and it stipulates some relationships or behaviour.

**Figure 26. Sample Feature Diagram [67]**

Feature models, on the other hand, are widely used to capture requirements by most domain engineering methods [112] [113]. Feature diagrams [114] allow us to model common and variable features and interdependencies between them. Figure 26 is an example of such feature models. As can be seen, a feature model is essentially a tree of features which can either be mandatory (a black circle) or optional (a white circle). Alternative features can also be denoted in such a model.

Further, domain models, as another type of conceptual model, are used to depict the structural elements and their conceptual constraints within the problem domain of interest. A domain model includes entities, attributes of them and relationships among them, plus constraints that govern the conceptual integrity of the structural elements. A domain model may include a number of views with each one focusing on a particular subject area or subset of the domain model. Figure 27 is a sample domain model which depicts classes and relationships.

Domain models are often made in Unified Modelling Language™ (UML®) [110], class diagram or a similar language (Ecore in Eclipse Modelling Framework (EMF) project [115], for instance). UML is a set of languages which can be used to specify, visualize, and document models of software systems. But it is possible to use UML for business modelling and modelling of other non-software systems too. UML 2.0 defines thirteen types of diagrams that are divided into three categories: six types of diagrams representing static application structure, three representing general types of behaviour, and four representing different aspects of interactions [110]:

- Structure diagrams include the class diagram, object diagram, component diagram, composite structure diagram, package diagram, and deployment diagram.
- Behaviour diagrams include the use case diagram (used by some methodologies during requirements gathering), activity diagram, and state machine diagram.
- Interaction diagrams (derived from the more general behaviour diagram) include the sequence diagram, communication diagram, timing diagram, and interaction overview diagram.
Among these diagrams, the use case diagram and class diagram may be the two most widely used ones. A use case is a description of a set or sequence of actions that a system performs. A use case is often rendered as an ellipse with solid lines including its name. A use case diagram shows a set of use cases and actors and their relationships [116]. A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class is rendered as a rectangle, usually including its name, attributes, and operations. A class diagram shows a set of classes, interfaces, and collaborations and their relationship. Usually, a use case diagram is for pure conceptual modelling, while a class diagram can be used for further generating parts of software codes as well.

2.4.1.2 Model Driven Development
Using models to design complex systems is a long standing tradition in traditional engineering. It is not possible to construct a complex bridge without first constructing system models. Models help us to understand a complex problem and its potential solutions through abstraction [117]. Software systems, often among the most complex engineering systems, try to apply and benefit from modelling techniques as well. This has become quite popular since the development of Unified Modelling Language (UML). However, the actual benefits have not been as large as expected. For historical reasons, models in software engineering were infrequently used, or used in a secondary role [117]. They have been thought of primarily as documentation artefacts, and their creation and usage have been peripheral to software development [108]. This flavour of model usage is called ‘model-based’ and its main disadvantage is that those models are only documents and therefore static, difficult to change, and considered to be an overhead [118]. If models merely serve as documentation, they are of limited value since documentations are easy to diverge from the implementation. This
narrow perspective has even led to “recurring and seemingly futile debates on the practical value of modelling in software development” [108].

However, Model Driven Engineering (MDE) advocates have pointed out that models can be used for more than just documentation during development [108]. In Model Driven (Software) Development (MDSD/ MDD), models, rather than computer programs, are the primary focus and the primary products of software development [117]. Models are assigned a central and active role, and they are at least as important as source code. They can even be considered equal to codes since codes can be automatically generated from models [117].

We can see that one of the key characteristics of MDD methods is their fundamental reliance on automation [117]. As pointed out by Bran Selic, an IBM Distinguished Engineer: “Software has the rare property that it allows us to directly evolve models into full-fledged implementations without changing the engineering medium, tools, or methods.” Modelling techniques leveraging this property can significantly reduce the complexities associated with handcrafting software [119]. Similarly, the formal methods community has attempted to leverage this property by systematically transforming declarative specifications to programs earlier [120]. [117] also thought that one of the two reasons as to why MDD is more useful today is because automation technologies have matured (the other reason is that industry-wide standards have emerged).

A model is an abstraction of some aspect of a system, and source code can be considered to be a model of how a system will behave when executed [108]. However, most model research is primarily concerned with the development and use of models other than source code. In contrast to source codes written in most popular programming languages, we express models using concepts that are much less bound to the underlying implementation technology while being much closer to the problem domain. This makes the models easier to specify, understand, and maintain; in some cases, it might even be possible for domain experts rather than computing technology specialists to produce systems. It also makes models less sensitive to the chosen computing technology and to evolutionary changes to that technology (the concept of platform-independent models is often closely connected to MDD) [117]. Therefore, MDD enables us to move the focus of work from programming to solution modelling [118].

With automation, such abstractions provided by models become more meaningful, and both the abstractions and the automation contribute to the benefits brought by MDD:

- “The model-driven approach has a potential to increase development productivity and quality by describing important aspects of a solution with more human-friendly abstractions and by generating common application fragments with templates” [118].
- “The term Model-Driven Engineering (MDE) is typically used to describe software development approaches in which abstract models of software systems are created and systematically transformed to concrete implementations” [108].

In addition to the increased productivity, in MDD, both the quality and maintainability of the software system also increases [118].
However, MDD may not be easy. MDD requires models to be *abstract and formal at the same time*. Creating abstractions is not straightforward. Defining models in a formal way (to make automation possible) requires several practical issues to be handled. [108] suggested two major challenges which architects of MDE modelling languages face:

- The abstraction challenge, meaning how to provide support for “creating and manipulating problem-level abstractions as first-class modelling elements in a language”, and
- The formality challenge, meaning how to find aspects of a modelling language’s semantics that need to be formalized and how to formalize the aspects.

There are two broad classes of models that MDD research may focus on: development models and runtime models [108]. Development models are software models at levels of abstraction above the code level. Examples include requirements, architecture, implementation, and deployment models. Runtime models, on the other hand, present views of some aspect of an executing system (abstractions of runtime phenomena). Most MDE research may focus on the development models, but a growing number of MDE researchers have started to explore runtime models. However, as the authors [108] pointed out, as MDE research matures, the classification may become dynamic. Development models may be used as runtime models, and runtime models may act as development models to evolve software systems. In this research, the focus is on the development models.

### 2.4.1.3 Domain Specific Modelling

Model driven development is domain specific [17]. This applies to modelling languages in the first place. As mentioned earlier, the key characteristics of MDD are providing higher level abstractions and automatic code generation. Being domain specific makes the modelling languages easier and more efficient. As pointed out by [17], “While seeking to raise the level of abstraction further, languages need to be better aware of the domain. Focusing on a narrow area of interest makes it possible to map a language closer to the actual problem and makes full code generation realistic—something that is difficult, if not impossible, to achieve with general-purpose modelling languages”. The authors indicated that, in many industrial cases and application areas (e.g. database design and user interface development) where MDD was applied, the modelling languages were domain specific. These domain-specific languages were often made in-house and less widely publicized. They might be more productive since they fit a narrower domain, and therefore are easier to create as they only need to satisfy in-house requirements.

In contrast, UML was developed to model all kinds of application domains, but it has not proven to be successful for MDD purposes [17]. UML has contributed greatly towards emphasizing the need to consider design prior to implementation. But UML does not raise the abstraction level above code concepts and does not support code generation adequately. As [17] stated, “we should keep in mind, however, that UML was originally set up not for automating development but for agreement on modelling concepts, their naming, and symbols”. The emphasis of the language was on specifying, visualizing, and documenting the artefacts rather than supporting developers in making the design decisions or automating software development.
As General Purpose Modelling Languages (GPMLs) like UML cannot support MDD in a desired way, another way to tackle the abstraction challenge is to extend the GPMLs. In [108], the authors introduced two schools of modelling languages that have emerged in the MDD community:

- The extensible GPML school: the abstraction challenge is tackled by providing a base general-purpose language with facilities to extend it with domain-specific abstractions.
- The Domain Specific Modelling Language (DSML) school: the challenge is tackled by defining domain specific languages using meta-meta modelling mechanisms.

The research of this thesis focuses on the DSML school. The focus of work in this area has been on providing tool support for engineering modelling languages [108]. Xactium, MetaCase, and Microsoft provide examples of such attempts. These will be further illustrated later on in the thesis.

In addition to modelling languages, code generators should be domain specific as well. They should be restricted to developing only certain kinds of applications on certain platforms since “it is obvious that we can’t have only one code generator for all software” [118]. In addition, having domain specific platforms would cause a significant complexity reduction of the code generator [118]. As a result, all the main constituent parts of MDD should be customized according to the specific domains that MDD solutions serve. The MDD’s nature of being domain specific may explain why researchers use the terminologies of MDD and Domain Specific Modelling (DSM) alternatively (which also occurs in this thesis). It also explains why the domain analysis is thought of as the starting point of MDSD[118].

2.4.2 Elements and Products

This section introduces constituent elements and products of model driven development.

2.4.2.1 Language

In [118], the authors formalized the definition of terminologies that are widely used in this area of research, and depicted important relationships between them. Figure 28 depicts the relationships between model, language and domain. In addition, what constitutes a domain specific language is also illustrated. Below, a brief summary of some important elements from [118] is presented.

The term domain, the starting point in MDSD, describes “a bounded field of interest or knowledge” [118]. Creating an ontology of a domain’s concepts can be useful to internalize and process this knowledge [118].

In the context of MDSD, the structure of a domain must be formalized in order to make any attempt at automation. This formalization takes places in the form of a meta-model which describes concepts that can be used to make models. The meta-model itself has a meta-model that defines the concepts available for meta-modelling. Consequently, there is a meta-meta model.
In a meta-model, both the abstract syntax and the static semantics of a language are defined. In addition to them, Domain Specific Language (DSL) (often used synonymously with the term modelling language) also possesses a concrete syntax and semantics. While the concrete syntax of a language such as Java specifies what a parser of the language accepts, the abstract syntax merely specifies what the language’s structure looks like. The static semantics of a language, on the other hand, determines its criteria for well-formedness. A typical example from the world of programming language is the rule that variables must be declared. Static semantics of UML can be defined formally using Object Constraint Language (OCL) expressions. These serve to detect modelling errors in terms of the formalized domain. The semantics of a DSL should be well-documented or intuitively clear to the modeller. If DSL adopts concepts form the problem space, it will be easier for the domain experts to recognize the ‘domain language’.

Formal models are the starting point for automated transformations. A formal model is obviously connected to the respective domain. It is formulated based on the DSL’s concrete syntax, and it is at least conceptually an instance of the meta-model (and technically as well in most cases). A formal model can be thought of as a sentence formulated in the DSL, and its meaning can be obtained from the DSL’s semantics.

2.4.2.2 Transformation
As mentioned previously, the key characteristic of MDD methods is their fundamental reliance on automation [117]. Models can be automatically transformed into implementation, and therefore models are the primary focus and products of developers. Various approaches are adopted in practice to synchronize models and codes. [17] enumerated five of these approaches, as Figure 29 presents: 1) no models at all; 2) models are used for design and are separated with codes completely; 3) models are used for visualizing and understanding existing codes; 4) models are used for code generation to some degree, but some of the codes are manually written and in turn used to update the models. This round-trip approach aims to automate the work in...
order to keep information located in two places the same and up-to-date. It only works when the formats are very similar and no information between translations is lost. Relatively few areas like structural specifications may be suitable for such an approach; 5) models are fully transformed into codes and models are the primary artefacts generated by the developers.

In [17], the authors argued that MDD approach should be applied whenever possible. And this kind of truly model-driven development should use automated transformation in a manner similar to how a compiler works. In addition, there may be other variants to the last approach. For instance, instead of generating models to executable codes, models are interpreted by model interpreters (engines) and directly executed without generating codes and running the codes. However, it is also possible that models are generated to some configuration file which configures the supporting platform or interpreter. Both the fifth approach (model transformations) and the variants listed above are typical MDD approaches. The research of this thesis is mainly concerned with general model transformations.

![Figure 29. Aligning Code and Models [17]](image)

It is not always straightforward and efficient to directly transform models to codes. In [118], the authors distinguished two ways to do this (see Figure 30): model to model (“model2model”) transformations, and model to platform (“model2platform”) transformations. In [119], it is stated that a software product can be created in part or in whole through one or more transformations (where at least one is a model2platform). The product can be an entire application or a component that would be used elsewhere. Such a product aggregates a platform, generates artefacts (from models), and sometimes also includes some non-generated artefacts like application-specific helper classes or manually programmed business logic (however, some researchers [17] proposed that MDD should aim to fully automate the code generation).

In addition, the authors [118] also introduced meta-models that are needed for the transformations. To make model transformation possible, source meta-models should always be available for the machines to understand the inputs (models). For model2model transformations, target meta-models are also needed. This is because the transformation creates another model and needs to know how to describe the mappings among the model constructs. In contrast, model2platform transformations ‘know the platform’ and generate artefacts that are based on it. Target meta-models are not needed, as usually we only deal with simple text replacements.
Similarly, [118] classified model transformation approaches as model-to-code approaches (corresponding to model-to-platform approaches above) and model-to-model approaches. Further, the authors illustrated more detailed mechanisms that transformations may be based on. For instance, model-to-code approaches may be visitor-based or template-based, while model-to-platform approaches may be of direct-manipulation, relational, graph transformation based, structure driven, or hybrid.

2.4.2.3 Platform

In the context of MDD, generated codes are often based on some reusable frameworks, super classes, common components, etc. [118] called this part a platform (see Figure 31). This platform is much more powerful than a ‘naked’ programming language or general technique platform. To highlight the domain specific feature of such a platform, some researchers have called it a domain specific framework or domain specific library. In [17], the authors proposed to introduce “this layer of code that sits between the generated code and the existing generic components and platform” to avoid competition and complexity in the generated code or in the generator. By doing so, some of the following profits can be gained:

- providing an interface for the generator,
- removing duplication from generated code,
- hiding platform details, and thus bypassing bugs and platform evolution, as well as extending a framework from one platform to many.

Therefore, involving such platforms that underlie generated codes can help in alleviating the platform complexity and expressing domain concepts more effectively.
2.4.2.4 Products

In [118], the authors proposed *domain architecture* to be the central concept in MDD. As Figure 32 shows, domain architecture aggregates the tools that are needed to transit models to the products: DSL, platform, and transformations. Domain architecture determines which concepts are formally supported (can be expressed with the given DSL), and how these are mapped to (can be realized on) the existing platform.

Further, [119] defined the terminology of *software system family* as “the set of all products that can be created with a certain domain architecture”. In other words, a software system family uses a specific domain architecture for its realization. On the other hand, a domain architecture is reusable for all the products/members within the corresponding software system family. To achieve this, the domain architecture must be “flexible enough to allow the expression of the differences (variabilities) between various products that make up the software system family” [118]. In addition, another kind of product set known as *product line* was defined as “a set of complementary single products” from a user’s perspective. That is, the products that make up a product line can constitute alternatives - they are applicable in different but related contexts or can complement each other’s content - and thus define a ‘suite’. In contrast to the software system family, the products within a product line do not have to share
any technical commonalities (like any part of the domain architecture). Therefore, a software system family may form the base of a product line, but not necessarily.

### 2.4.3 Process, Tools, and Challenges

In this section, the general process of model driven development is presented. In addition, tools and challenges are also introduced. MDA is introduced briefly, as the most prominent of MDD approaches.

#### 2.4.3.1 Process

Usually, DSM happens to a familiar domain instead of a new one [17]. Roberts and Johnson [121] described a similar recurring pattern for software development automation, which can be regarded as the initial form of the DSM development process. This process consists of three phases: 1) identify a set of reusable abstractions and document a set of patterns for using them; 2) develop a runtime, such as a framework, to codify the abstractions and patterns; 3) define the languages and build the tools to support the runtime (such tools include editors, compilers and debuggers which automate the assembly process). Similarly, [122] identified five phases of DSL development which include decision, analysis, design, implementation, and deployment. Similarly, [123] indicated that developing a DSM technology should combine both the development of domain specific modelling languages and the development of transformation engines and generators. In addition to DSM development, a DSM process should also include DSM usage. [67], inspired by [124] and [125], subdivided the development phase to analysis and design phases, and enumerated nine tasks within a DSM approach, as shown below.

- **Analysis phase:**
  - 1) identify the (problem) domain;
  - 2) collect relevant domain knowledge;
  - 3) cluster the knowledge in semantic notions and operations;
  - 4) design a DSL which describes the domain applications;

- **Implementation phase:**
  - 5) construct a framework (i.e. a set of domain specific libraries) which implements the semantic notions;
  - 6) construct a compiler (i.e. a code generator) to translate DSL programs to a sequence of calls to the framework;
  - 7) construct a visual editor to graphically manipulate the DSL if visualized modelling is needed;
  - 8) create semantic validators to identify design errors if necessary;

- **Use phase:**
  - 9) write DSL programs and compile them (to generate codes).

#### 2.4.3.2 Tools

Corresponding to the two schools of modelling languages introduced earlier, there are two classes of DSL tools: generic tools such as UML tools configured by a profile, and custom-made DSL specific tools [17]. While the generic tools have been thought to be not quite helpful for DSM (the most representative generic tools, MDA and
UML, will be introduced later), most of the current approaches are about the custom-made language specific tools. According to [17], such tools can be built at six levels:

1. build from scratch;
2. build based on frameworks;
3. construct a meta-model, generate a skeleton of the tools over a framework, then add more code;
4. construct a meta-model, and generate the full code of the tools over a framework;
5. construct a meta-model, output configuration data for a generic tool;
6. build within an integrated meta-modelling and modelling environment.

To enhance the creation of these tools, some “meta” tools have been provided by various organizations and companies. They are called language-based tools, language-oriented tools or language-workbenches. To avoid ambiguity (with “language specific tools”), the term language workbenches will be used further for these meta-tools.

Language workbenches provide the means (and often with an IDE) to facilitate the DSM tasks and automate a large portion of the overall programming work like language definition, tool-chain creation, and assembly process [67]. In [125], as summarized by [118], the authors argued that the “fate of DSLs basically depends on how easy it is to build new languages and integrate them into everyday development environment.” Thus language workbenches are considered to be the ‘killer app’ for DSLs. [17] reviewed several DSM environments (with language workbenches built in) like MetaEdit+, GME, MS DSL Tools, and Eclipse Modeling Project. Among them MetaEdit was thought to be the most mature, sophisticated, and also the most widely known (recognized by over 50% of lead developers, as shown by an independent survey by MediaDev in Europe in 2006). The authors [17] analysed these tools according to six levels, indicating a level for each one. With version 4.5 released in 2006, MetaEdit+ was thought to be at level 6, with meta-modelling and modelling supported in a same integrated environment. GME was at level 5. By using GME, a meta-model is turned into a binary file configuring a second instance of GME. MS DSL Tools and Eclipse EMP (EMF and GMF 2006) were at level 3.

A software factory is a similar IDE configured for the efficient development of applications in a specific domain. The configured IDE makes use of languages, models, frameworks and patterns as simply as possible. Thus “the concept of Software Factories is the industrialization of software development ‘from craftsmanship to manufacturing’”, and software factories are described by some as “doing product lines the Microsoft way” [118]. However, although DSM is an important ingredient, software factories are much wider in scope than ‘just’ DSM, as they look at the complete product-line development. Similarly, [126] indicated that the idea of software factories is basically a clever combination of three key concepts: MDD (concept of abstraction), component-based development (concept of granularity) and software product lines (concept of specificity).

However, this situation has changed to some extent as tools have continued to develop throughout the years. To our best knowledge, with the contributions from Xtext and Sirius projects, Eclipse can work as quite a mature platform and approach level 6 as well. Also, according to [127], over the past few years, several new initiatives “have surfaced in the area of creating so called language workbenches – aiming at facilitat-
ing the definition and use of DSLs and code generation. Each of these has its own strengths and weaknesses, and none is ‘the best’ for every purpose”. And a recurring workshop (Language Workbench Challenge) has been organized “to help people get more insight into what is available and possible [127]”.

2.4.3.3 MDA and UML

MDA has been one of the most prominent and well known MDSD approaches for almost ten years. In [117], the author indicated that the emergence of standards (e.g. those in support of MDA) is one of the key factors in making MDD more useful (the other key factor is the maturation of automation technologies). In 2001, MDA was voted by Object Management Group (OMG) members to be established as the base architecture for other standards of the organization [110]. MDA proposed software development starts with a Platform-Independent Model (PIM) where business functionalities and behaviours are modelled. Then the PIM is firstly converted to a Platform-Specific Model (PSM) and secondly to a coded application on a middleware platform (such as web services XML/SOAP, and CORBA). There are many OMG’s modelling specifications which support the MDA, as Figure 33 shows. OMG Task Forces organized around industries use the MDA to standardize their domain specific facilities. MDA development tools, like the transformations, are expected to be available from a number of vendors. Related standards include:

- The Meta-Object Facility (MOF): MDA requires that all the models, whether PIMs or PSMs, should be expressed in a MOF-based language;
- The Unified Modeling Language (UML, now at Version 2.0): for many specifications, PIMs and PSMs will be defined in UML. But this use of UML is not mandatory;
- The UML Profiles: UML Profiles tailor the language to specific areas of computing or specific platforms;
- The XML Metadata Interchange (XMI): for MOF-based (including UML) meta-models and models, XMI defines an XML-based interchange format.
- The Common Warehouse Metamodel (CWM): the CWM standardizes a meta-model which enables data mining across database boundaries at an enterprise;
- CORBA: MDA targets every middleware platform. CORBA can be thought of as a target platform which is independent of programming language, operating system, and vendor.

As summarized by [118], MDA can be thought of as a specialization of MDSD with characteristics like:

- MOF is used as its meta-meta model;
- UML profiles are used as a concrete syntax for a DSL. Accordingly, the static semantics are specified by OCL expressions;
- MDA recommends that transformations are carried out in several steps between models (PIM or PSM), however it does not prohibit a direct transformation from a PIM to code;
- MDA defines Platform Description Models (PDM) as a meta-model for platforms. By describing platforms as a PDM, even the final transformations connecting with the platforms will also be possible to describe;
- OMG’s QVT is used as the standardized transformation language;
- Executable UML models are expected by many MDA representatives.
Despite the grand expectations of OMG, researchers have indicated that MDA did not achieve large success during the past decade mainly due to heaviness and bad support. In 2006, [9] summarized some significant issues that MDD (mainly considering MDA) might have raised in practice: 1) **Redundancy.** As [9] pointed out, MDD requires multiple models representing different views or abstraction levels of the same concept, and this will lead to duplicate work and consistency management; 2) **Rampant round-trip problems.** Multiple models and abstraction levels indicate increasingly complex relationships among the models. This issue becomes worse when the round-trip problem can’t be addressed automatically and when changes occur in artefacts at lower abstraction levels such as code; 3) **Moving complexity rather than reducing it.** Some degree of software complexity is inherent in the problems, and simply reducing visible complexity and moving complexity elsewhere may cause big issues when the full development cycle is considered; 4) **More expertise required.** Although MDD aids in the communication between different groups of participants, it also implies that the different groups cannot be expert in only their specific domains. This is because artefacts generated at any stage of the overall process can impact other artefacts generated at any stage.

In addition, regarding the modelling languages, the authors in [9] indicated that the standardization of modelling notations is important for achieving MDD. But the UML 2.0 standard has had serious problems which has impeded the adoption of MDD. Firstly, UML is enormous and unwieldy in attempting to address too many disparate needs; Secondly, MOF enables UML to be extended “almost arbitrarily”, thus some of UML 2.0 constructs like use cases are nearly “semantics-free”. This has complicated using UML extensions, reduced their expressive power, and limited the ability of vendors to provide reliable and consistent tools. Lastly, there are the challenges of automatically generating executable code from high level descriptions.
In 2011, Jean Bézivin gave a talk to the second Brazilian Workshop on Model Driven Software Development entitled: “Why did MDE miss the boat?” [130]. Bézivin indicated that the best known instantiation of MDE is the MDA, but the deployment of MDE seems to have reached a standstill: “Models have failed, at least temporarily”. “In spite of the immense hopes that greeted the MDA initial proposal as a possible way to regenerate the entire software engineering practices, we must recognize today that its impact is rather limited and its perspectives quite confined”. And similarly, Darius Silingas reviewed multiple real world MDA cases and suggested some reasons as to why MDA did not succeed [128] (see Figure 34).

2.4.3.4 Challenges

MDA, as the most visible form of MDE, has failed to reach wide adoption. MDE in general, however, is not easy to start with either. Despite the potential for yielding major productivity and reliability benefits, MDD may still be rejected if it is not performed in an acceptable form. In [117], the author proposed that a model, as an essential part of engineering, helps us to reduce risk by better understanding a problem and its potential solutions before undertaking the expense and effort of the full implementation. Such models must possess five key characteristics to be useful and effective:

- Abstraction. Models help us understand the essence of a problem more easily by removing or hiding details that are irrelevant from a certain perspective;
- Understandability. Models present the remains (after abstracting away the details) in a form that most directly appeals to our intuition. This is a direct function of the modelling form’s expressiveness;
- Accuracy. Models provide a “true-to-life” representation of the system’s features of interest;
- Predictiveness. Model can be used to predict interesting but nonobvious properties of the system, either through experimentation or formal analysis. This is
greatly dependent on the modelling form and the accuracy of the models;

- Inexpensiveness. Models should be significantly cheaper to construct and analyse than the system.

In addition, the author in [117] indicated that models are of limited value without fully exploiting the potential for automation. In contrast to attempts which were limited to diagramming support and skeletal (and fragments) code generation, *automatically generating complete programs* as well as *automatically verifying models* were thought necessary to attain MDD’s full benefits. Since maintaining models and keeping models and codes consistent usually requires scarce and expensive resources, models are often abandoned once the codes have been generated.

What is more, the *model driven development approaches* face overall pragmatic challenges that prevent them from being accepted by more software practitioners. In particular, five issues were highlighted by the author [117]:

- Model-level observability. When an error is detected in the generated program, it should be easy to find the place in the model that must be fixed. As an analogy for traditional programming languages, we expect compilers at compile time or debuggers at run time to report errors. This need is greater for models because of the semantic gap between high-level model abstractions and low-level code implementations. Other related facilities are inverse transformation “templates”, model mergers, and model difference tools;

- Model executability. Executable models provide us with early direct experience with the modelled system;

- Efficiency of generated code. Efficiency (performance and memory utilization) is not an issue for the vast majority of applications. For more critical cases, the MDD system should allow for “seamless and overhead-free” embedding of the critical elements;

- Scalability. Tools and methods must scale up to meet large-scale industrial applications. It should be possible for hundreds of developers to work on hundreds of parts of a model that are different but related. Compilation time (full-system compilation time and more important turnaround time for small incremental changes) and system size are important. However, the overhead of automatic code generation is almost negligible compared to the usual compilation. The system size seems not to be a problem according to the experience of the largest systems to date;

- Integration with legacy environments (including process) and systems (like tools and libraries). The author [117] indicated that a prudent and practical way to introduce MDD into an existing production environment is to apply it to a smaller-scale project like a lower-profile extension to the legacy system. This implies that not only the software, but also the development process and environment must be integrated into the legacy process and environment. This is to mitigate risks and also to leverage previous investments into the new process and environment. The integration should “work out of the box” and not require custom glue code and tool expertise in general.

Similarly, [131] proposed that the two main quality criteria for models to be used in MDE are transformability and maintainability. Solheim and Neple decompose these criteria into several other criteria. The quality criteria for transformability are shown in Table 8, while the maintainability criteria are shown in Table 9.
Table 8. Transformability Quality Criteria for Models [131]

<table>
<thead>
<tr>
<th>Quality criterion</th>
<th>Type of quality</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>Semantic</td>
<td>The model contains all statements that are correct and relevant about the domain. This can be checked against the ontological meta model.</td>
</tr>
<tr>
<td>Well-formedness</td>
<td>Syntactic</td>
<td>The model complies with its language definition. This can be checked using the linguistic meta model.</td>
</tr>
<tr>
<td>Precision</td>
<td>Technical pragmatic</td>
<td>The model is sufficiently accurate and detailed for a particular automatic transformation.</td>
</tr>
<tr>
<td>Relevance</td>
<td>Technical pragmatic</td>
<td>The model contains only the statements necessary for a particular transformation.</td>
</tr>
</tbody>
</table>

Table 9. Maintainability Quality Criteria for Models [131]

<table>
<thead>
<tr>
<th>Quality criterion</th>
<th>Type of quality</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traceability</td>
<td>Technical pragmatic</td>
<td>The model’s elements can be traced backward to their origin (requirements), and forward to their result (another model or program code).</td>
</tr>
<tr>
<td>Well-designedness</td>
<td>Syntactic</td>
<td>The model has a tidy design, making it understandable by humans and transformable to an understandable and tidy result.</td>
</tr>
</tbody>
</table>

2.5 Models for Game Design

This section introduces model practices for both high level game design and low level or complex game logic design. It also investigates how UML, as the best known universal modelling language, has been applied in the game design field.

2.5.1 Models for High Level Game Design

Flow charts have been widely used for high level game design. In 1999, Lewinski [132] described the use of flowcharts. Such flowcharts depicted the scene or mission in a computer game (represented by an ellipse) flowing to the next scene or mission (by means of an arrow). Similarly, Gold [133] suggested a diagrammatical representation of the route through a complete game level as a topological map or graph, with nodes representing scenes and edges representing the transitions between them. The author advocated that such maps be used for scenic (non-interactive) objects. [134] presented a flowboard as an example of such modelling practices (see Figure 35).
However, some researchers realized the limitation of such flowcharts. Rouse [136] suggested that flowcharts were not actually all that useful in the game design process, other than to serve as the communication function. Instead, Rouse supported the use of storyboards (as used in film and television production) for mock-ups of how a game will appear to the player. In 2002, Onder described the use of story-beat diagrams, which are a collection of ovals and arrows that help show the flow (or alternate flow) of a computer game story at a high level. For each scene in the story-beat diagram, the scene location, scene description, the object and cast members in the scene, and an event table (that includes action / result entries that describe what happens in response to each player inside the scene) need to be recorded [135]. Figure 36 presents an example of such a story-beat diagram.

Some other kinds of models have also been utilized. [68] suggested the use of token interaction matrices (see Figure 37). A token interaction matrix is a chart of all the interactions that take place in (a segment of) a game. A token is defined as a discrete game element that is (directly or indirectly) manipulated by the game player.
2.5.2 Models for Low Level and Complex Game Design

In addition to high level game design, researchers have also tried to find the proper means to aid their design activities regarding lower level and usually more complex game logics. Various kinds of Statechart or Finite State Machine (FSM) diagrams might be the most representative tools available. As early as in 2002, [137] advocated State Transition Diagrams for game loops to assist game programming activities, and for decision trees to develop artificial intelligence coding for computer games. [68] presented an example of similar Statechart Diagrams, as Figure 38 shows. In the diagram, a square is used to represent an individual state of a game object. A circle indicates an event. And lines with a colored dot show the transition between states and the incoming event that triggers a transition.

Further, Grünvogel [138] proposed the use of a mathematical model (Abstract State Machines) to precisely describe games. In such a model, games were modelled as a system of objects whose state was changed by the players and other game objects. This precise specification method completely removed the ambiguity of natural language. However, game designers without a solid knowledge of the mathematical notation would find it difficult to express their designs. FSM was used in modelling game
AIs [139], and some other game aspects such as character behaviour, game flow and game scenario. In 2007, [139] used a variant of *Rhapsody Statecharts* [140] as a combination of statecharts and class diagrams (see Figure 39). In such a chart, class diagrams were used to represent the properties and behaviour of game objects, and statecharts were accompanied to change or switch the state of a game object as certain conditions were met.

![Figure 39. Rhapsody Statecharts of EnemyTracker AI Component [139]](image)

In addition to statechart variants, *Petri nets* and their adaptations are often used. For instance, [142] used the mathematical modelling language (*Petri nets*) to specify rules in iconic diagrams. In such a model, nodes and links are used to represent concrete game design concepts like atomic transitions, texts, resource sources, storage, sinks and flows. However, the final diagram specification is difficult to understand and to scale. [141] proposed another approach to address the need for modelling spatial-temporal relationships. The author proposed to use an adaptation of Petri nets and generalized graphs which are called *hypergraphs* (see Figure 40).

![Figure 40. Partial Map of Silent Hill 2 Represented in Hypergraph [141]](image)

In contrast to these design vocabulary approaches, the Machinations framework [143] has tried to do more by visualizing the structures in game mechanics that create emergent gameplay. A tool set was constructed to allow game designers to interact with
the structures that contribute to the game quality more directly. Figure 41 presents thenotations of such a Game Economy Diagram.

Figure 41. 'Game Economy' Diagrams [143]

2.5.3 UML for Game Design

As a generic modelling language, UML has been used in various domains including the computer game domain. In [144], the author illustrated how a range of UML diagrams (use case diagrams and class diagrams) were used to effectively document the design of a game software. Figure 42 shows an example use case diagram representing a player’s interaction and an example class diagram representing game objects and static structure. Similarly, [145] represented player interaction with use case diagrams (as Figure 43 shows) to identify all possible interactions in the game before representing it in a class diagram.

In addition to use case diagrams and class diagrams, [146] proposed using a larger subset of UML for developing computer games. Such diagrams include component diagrams representing the dependencies of the source code files, activity diagrams for program execution flow, and sequence diagrams for the interaction among program objects. Similarly, [147] used class diagrams and state transition diagrams (or activity diagrams as an alternative) to represent game structure and game behaviour. They also used object diagrams to examine the run-time behaviour and sequence diagrams to extend the modelling of behaviour between game objects. What is more, a collection of classes can be ordered into a high level view with component diagrams, while
physical arrangement of the game can be represented in a *deployment diagram*. Figure 44 presents an example of the structure diagram.

Some researchers have tried to extend or adapt standard UML to better meet their requirements. [104] demonstrated how they extended and adapted *use case diagrams* by incorporating aspects of decision trees to provide a means to design detailed computer game-flows. Figure 45 shows what such extended use-case diagrams look like. These kinds of use-case diagrams were designed not only for game programmers, but also for other professionals like story, level, and character designers, 3-D modellers, artists, animators, and musicians. On the other hand, [148] used *class diagrams* and an
extended version of the state diagram as a basis for modelling flow and in-game components (as Figure 46 shows).

![Diagram of a Computer Game-Flow Diagram](image.png)

Figure 45. Example of a Computer Game-Flow Diagram [104]

### 2.6 Models for Game Creation

Model driven game development has followed various directions. These directions are grouped into four approaches: product line approaches, MDA approaches, in-house approaches, and language oriented approaches. **Product line approaches** usually start with a feature model describing the mandatory and optional parts for the target system. Although not mandatory, DSLs are often used for product line approaches. **MDA approaches** have tried to keep pace with the OMG MDA architecture by isolating PIMs and PSMs, creating model transformations, and utilizing OMG standards (like UML, OCL, ATL, MOF, etc.). In contrast to product line approaches and MDA approaches, in-house approaches do not follow a similar architecture or process. Instead, they use a self-defined architecture or process based on a general or often self-developed platform to implement a model driven approach within some domains. **Language-oriented approaches** have recently become more utilized. These approaches encompass DSLs and corresponding code generators as the main artefacts that need to be developed. Language workbench tools are often utilized to automate most of the general work.

#### 2.6.1 Product Line Approaches

Product lines were once used quite widely. [149] conducted an exploratory study regarding the application of product lines in a mobile games domain company. The authors raised and validated hypotheses such as “Companies from specific domains like the mobile games domain already adopt a SPL approach without being aware of that”, “SPL adoption for SMEs in the Mobile game industry is especially costly”, “SPL is a promising business strategy for mobile games companies”, “The extractive adoption strategy, opposed to the proactive approach, is more suitable for SMEs in the mobile games domain” and “Mobile games is a complex domain”. On the other hand, [150] presented a practical approach to implement core assets in a SPL applied to the mobile games domain. A case study based on three different mobile games was described.
in detail. The results of the case study showed that this practical approach can be suitable for the mobile domain. Figure 47 shows the overview of the process [150].

Figure 46. Extended State Diagram that Models Partial Flow for "Thief robbing a lady" Scenario [148]

Product lines development usually requires a feature model at the beginning. Figure 48 presents a feature diagram for a mobile RPG domain [151]. In [151], an industrial mobile device project was reported where XVCL based solutions were used to build and manage the RPG product line architecture (RPG-PLA). XVCL (XML-based Variant Configuration Language) is a meta programming language, tool and method for building reusable software assets for any textual contents [152]. Similarly, in [153], the authors proposed the Object Oriented Feature Modeling (OOFM) [154] usage in the digital games domain. Distinct game features provided by NESI [155] and GDS [156] models (see Figure 49) were managed in a specialized OOFM structure. In addition, a feature-based environment was provided to integrate and adapt representative game features in different game engines.

Figure 47. Process Overview of SPL Domain Implementation Phase Applied to a Mobile Domain [150]
Although DSM is not a mandatory part of product line development, some researchers have presented approaches that include DSM. Examples include [157] and [126]. Their research will be illustrated in more detail later, as it focuses on DSL development. [158] presented a visual DSL for a Role-Playing Game (RPG) product line as well, but the approach involved several model transformation tasks, which was not typical. Thus it will be introduced in section 2.6.3 under in-house approaches.

2.6.2 MDA Approaches

As MDA emerged as the domain standard, many researchers tried to base their approaches on MDA in order to benefit from it. In [147] and [159], such an approach was introduced for 2D platform games (using Microsoft’s XNA). Within this approach (see Figure 50), a framework was used which was composed of two PIMs, one PSM to map game actions to hardware controls, and a transformation tool to translate these models into C++ source code compliant with the game engine. The two PIMs consisted of one structure diagram using a UML class diagram extended with stereotypes, and one behaviour diagram using a UML state transition diagram (see Figure 51). By defining control mapping, the game behaviour was separated from the controllers such as the joystick, keyboard, etc. Model to code transformation in MOFScript generated the majority of the game codes, while the programmer needed to manually code the rest (e.g. the AI). The authors concluded that the approach enhances productivity because 93% - 94% of the total codes could be automatically generated. However, the cost of modelling and tools construction was not factored in. Also, due to the limitation of MDA itself, the authors realized that the abstraction level was not high enough. In addition, PIMs were not translated into PSMs; the authors planned to do so in future work.

In other research by the same authors [160], a graphical DSL (referred to as PIM) for videogame gameplay specification was proposed, regarding the social context, structure, and rule sets of video games. Both meta-model and concrete syntax were given in detail. The authors expected that the DSL could work as part of a unified game DSL which also consisted of all other aspects like graphical interface, audio design, AI, game storytelling, level design, etc. So their work could be a first step in the overall model-driven game development methodology. However, as criticized by [15], in order to meet all the requirements from every individual game, the language will have to become more generic and less precise. Thus, the end result might not be quite prac-
tical. What is more, parts regarding the language, such as tool support (e.g. editor, compiler, and generator), were not described in [160].

Similarly, in [161], a solution was proposed for education games UI development. The approach defined meta-models in UML, and PIM and PSM models in XML. JavaSWIXml was used to produce a concrete java user interface based on SWING user controls from the XML files. Transformations from PIMs to PSMs were implemented with XSL Transformation (a language for transforming XML documents into other XML documents) [162]. And in [163] and [164], the authors described the potential for developing serious games using MDE approaches. In the paper, a grant model-driven serious game framework consisting of nine modules was described: 1) User Interfaces (UI), 2) Models, 3) MDE Tools, 4) Components Library, 5) Code Templates, 6) Artefacts, 7) Technology Platform, 8) Operating Platform and 9) Software. However, this work was not followed up with much concrete progress.
MDA approaches were also presented for developing board games [165]. In the approach, concepts including GameEngine, GameElement, Player, Event, Action, GameState, Goal, Sub-Goal, Non-MovableElement, MovableElement, Rules, and the meta-model were covered by the language represented in UML and UML stereotypes. OCL was used to define the static semantics. In addition, a UML class diagram was used for the concrete syntax definition. ATL was proposed to perform the M2M transformations between their models and the models instantiated from GameDSL (a meta-model for video games) to improve the interoperability. As for Model to Text (M2T) transformations, MOF script was used to generate class structures, and the OpenArchitectureWare (oAW) tool was used to generate textual game representations. The oAW tool is based on the EMF Ecore model, and the authors used xpand to translate the XML model which conforms to the meta-model defined in an XML schema (XMD). In contrast, [166] proposed MDA approaches for strategic based computer games. The authors argued that applying MDA is often very complex because the process of applying tools to automate translation between PIMs and PSMs are problematic. The authors proposed to build PIM models (in UML with OCL annotations), export them to files (in XMI), and to run them to avoid possible problems.

2.6.3 In-House Approaches

In [139], the authors showed how they built the tank model in their AToM3 visual meta-modelling and model transformation environment. In the environment, the model was compiled into C++ code with their own custom-built Statechart compiler (a variant of Rhapsody Statecharts). The AI codes (Figure 52 shows the components) were inserted into the Tank Wars game main game loop (in the AI function) by the custom tools.
In [11], a textual, **XML-based markup language (<e-Game> language)** was introduced. The resulting scripts could be used individually or together with a storyboard with the scripts marked on it. The approach was implemented in the <e-Game> project where a development process was proposed (see Figure 53). As the process shows, the resulting marked document, along with the art assets, could be interpreted by the <e-Game> engine. And, the executable videogame could be automatically produced.

![Figure 53. The <e-Game> Development Process [11]](image)

In [167], the authors indicated that traditional scripts describing interactions between a player character (PC) and objects in computer games are usually too complex for game story authors without programming expertise to write. Instead of writing such scripts in a procedural programming language, the authors proposed an approach (**Adaptive Programming**) for game authors (non-programmers) to generate the necessary scripts using a three-step process. First, the authors use a generative pattern to create a high-level description of a common game scenario. Second, the authors use a standard set of adaptation operations to customize the description, according to the particular circumstances. Third, the authors press a button to automatically generate the final scripting codes. Three studies were made and in total 56 game story authors used this process and their ScriptEase tool to construct Neverwinter Nights game stories. The authors advocated this kind of adaptive programming as an alternative to current constructive programming techniques for future game story scripting, as well as other domains.

In [12], an **expressive scripting language (SGL)** and system for developing data-driven games was proposed. SGL allows designers to modify the schema, scripts and preference files, so that gameplay can be easily changed without altering the core engine. The core of the SGL system is a compiler which translates SGL scripts into relational algebra and produces a set of C# classes used for rendering graphics. An SGL game consists of four components, as Figure 54 shows: schema definition for all the object and character types, scripts which can access the attributes in the schema, starting configuration, and interface description specifying how commands are mapped to Graphical User Interface (GUI) buttons. These components are specified by game
designers. SGL scripts are compiled with the schema definitions into an intermediate file which will be executed by the game engine.

![Figure 54. The SGL Workflow [12]](image)

In [88], the authors proposed an extension to the component based game software architecture, substituting blueprints with a domain model written in the OWL Web Ontology Language. The domain model included a description of the entities, their attributes and components, along with the messages they exchange (see Figure 55). While entity, component and message ontologies were represented as class hierarchies in OWL, the attribute ontology is represented as an OWL data property hierarchy. OWL is an expressive general-purpose language to represent knowledge, but it is difficult to use and offers different ways to define the same thing. Thus a tool named Rosette (see Figure 56) was proposed by the authors to enable programmers to design entities, components, messages and attributes in a visual way without writing a single line in OWL. The tool also connects to a reasoning engine which provides feedback to the end user.

![Figure 55. OWL Definitions for Entities, Components, Messages and Attributes [88]](image)

In [158], a visual DSL was presented for Role-Playing Game (RPG) product lines. In the project, the execution platform was Corona SDK, while the analysis platform was ALPiNA-OCL[168] for static analysis, and ALPiNA [169] as model checker [170] to check the invariants in Algebraic Petri net (APN) [171] models (see Figure 57). The APN models were expressed with Ecore format. The DSL was based on Ecore metamodel, and a graphical editor was constructed based on GMF/EuGENia. The M2M
transformations were done by means of the Atlas Transformation Language (ATL), and the M2C transformations utilized the template tool XPand.

Figure 56. Rosette, the Game Entity Editor [88]

Figure 57. Transforming a Source Model to a Target Framework with Model Verification [158]

2.6.4 Language Oriented Approaches

[67] was the first scientific paper found where model driven techniques were used for game development. In the paper, SharpLudus Game Software Factory was introduced for 2D adventure games. The framework consisted of two visual domain-specific modelling languages, semantic validators performing checks on the model, and code generators built on top of the Microsoft Visual Studio IDE to generate C# code for the associated game engine developed using DirectX technology.
The main DSL is the **SharpLudus Game Modeling Language (SLGML)**. Its top-level concepts and syntax are shown in Figure 58. The second DSL is the **HUD Creation DSL**, which can be used to specify how useful game information (score, remaining lives, hit points, etc.) would be presented by means of a heads-up display. With a detailed game specification created by these two DSLs, VSTO [172] technology can be used to create a User Manual skeleton, while code generators associated with the DSLs can be used to automatically create the majority of the game code. The generator receives a SLGML diagram as input and generates C# classes as output for audio components, sprites, entity classes, entity instances, states, the main game class, etc. Developers need to manually write some code to complete the program. Figure 59 presents the complete modelling experience for SLGML, which was hosted in the Visual Studio .NET IDE [173].

![Figure 58. Top-level SLGML Concepts and Syntax [67]](image)

![Figure 59. Complete SLGML Modelling Experience [67]](image)
In [126], the authors also advocated the use of a *software factory* as a clever combination of software product lines (concept of specificity), component-based development (concept of granularity), and model-driven development (concept of abstraction). *DiaMeta* was used to generate editors, functioning as a language workbench. A two-stage workflow was illustrated, as Figure 60 shows. In the workflow, product line developers needed to specify an *EMF model* representing the DSL definition, and an editor specification for the Visual DSL (VDSL) definition to generate the level editor (basically a visual language editor). After that, product developers used the factory to model games. Both the level editor and an additional textual DSL editor (provided by DiaMeta) may be used. Some codes need to be manually implemented. The resulting game factory is shown in Figure 61.

**Figure 61. The Pacman Factory [126]**

In [13], *Eberos Game Modelling Language 2D (Eberos GML2D)* was proposed for 2D games modelling. The DSL can be used to describe sprites and animations, entities, logic, collision detection and game controllers. A graphical editor was created by *Microsoft DSL Tools* [174] and a code generator was created to transform Eberos GML2D models into Microsoft XNA code. An Eberos GML2D model is shown in Figure 62. To evaluate the approach, two games were modelled and compared with writing the game from scratch. The results showed 86.4% savings on programming effort and 82.3% savings on programming time.
In [14], a DSL was proposed to describe structure and behaviour for simple, and data-centric mobile applications. The core concepts included application structure, entities, content providers, and views. The tool support covered the whole development cycle (for the iPhone development platform) including static analysis, compiler, simulator integration, etc. Xtext was used for DSL design and validation rules implementation. The code generator was developed with Xpand, which evaluated templates and produced text from EMF objects coming from Xtext. The Eclipse platform was leveraged as an IDE, to implement incremental code generation, code compilation of the generated codes, and invocation of the iPhone simulator.

Similarly, in [15], the authors used Xtext and Xpand (as well as the ANTLR9 parser) as the main tools to implement their DSL. The DSL included basic concepts to describe visible structures of 2D Point&Click Adventure games, such as room, door, character, object, action-verb, context information, and gameplay related entities in-
cluding Task, Goal, Constraint, Action, and Dialogue. The DSL was introduced as an illustration of a general language engineering workflow (see Figure 63). In the workflow four tiers were included. The three lower tiers represented the actual language engineering process as part of the game design. The top level tier was the interface to the implementation phase. The authors also indicated that in an iterative process, the DSL may be refined to fit the needs as game design is refined. On the other hand, the DSL may be used as a tool to refine the game design.

Figure 64. Architecture of PULP Tool Chain with Media and Script Artefacts [175]

[175] used Xtext toolkits based on the Eclipse platform as well. But the M2T transformation was based on Xtend2. The paper introduced work in progress about a textual DSL named PULP to tying content (like images and media files) by modelling the dynamic behaviour, movements, and control flow. A PULP script contains some global constants, screens, screen objects, and associated interactions expressed as action chains. Figure 64 shows the architecture of the tool chain involved.

2.7 Summary

This chapter studied background knowledge in game domains related to the research of this thesis. The challenges of modern computer game development were presented. The focus of this thesis has mainly been on tackling the challenges of improving software quality and productivity (by utilizing MDD approaches). However, there is no commonly used definition for pervasive games. And available domain vocabularies may not serve well for model driven use. These constitute challenges to the research of this thesis.

In this chapter the background of general model driven techniques was also presented. Models provide abstractions of target systems, while MDD provides code automation based on formal models. MDD is not easy, despite the benefits it may have, such as decreasing complexity, shortening the development cycle and improving software quality. The elements and process of MDD were introduced, as were the challenges of involving MDD. Among the challenges, the model quality issue as well as the cost issue may be the most vital to the success of applying MDD. Whatever domain MDD is to be applied in, these issues must be handled well.

Finally, this chapter surveyed the roadmap and areas where models were applied in game design and game development. Although model driven game development includes models for game design, MDSD imposes extra (and higher) requirements for the models than pure game design does. However, the state-of-the-art MDGD did not address these requirements very well. For instance, how the domain model was in-
vented and how its quality was ensured has not been clearly stated. Also, most of the scientific papers found did not address the peculiarities of the game development process. The research of this thesis addresses general MDSD challenges (mainly about the model quality and the cost) when applying MDSD in the game domain (making adaptations according to the domain requirements). The focus is on the domain analysis part (for the quality issue) and procedural part (for the cost issue). According to the requirements raised by the domain analysis, the lack of properly defined domain vocabularies is one of the biggest challenges to this research. The different directions that MDGD practices have followed were also reviewed. Currently the main trend is language oriented as such approaches are compact, highly automated, and agile. The research of this thesis is in keeping with this trend.

The above background knowledge provided the theoretical base for the contributions of this thesis, and the challenges described in this chapter led to the construction of the research questions. These will be illustrated in detail in Chapter 3.
Chapter 3 Research Method

This chapter first summarizes the challenges found in Chapter 2. Then, the research questions are described with a detailed rationale for why they were chosen and why they were considered to be interesting. The main research approach used (Design Science) towards the research contributions is outlined. Lastly, the research process is introduced.

3.1 Challenges

Despite the various benefits that can be expected from applying MDSD, such as shortened development cycle and improved software quality, the application of MDSD in the game domain does not yet appear to be mature and successful. Before any contributions can be made to this area, we must be clear about the main challenges that exist. A literature review was conducted regarding model driven game development practices (as presented in Chapter 2). Only a limited number of scientific papers (less than thirty) were found, as this is a new domain. In addition, this thesis has identified four main challenges with respect to the domain analysis part and the overall procedure of Domain Specific Modelling (DSM) approaches:

1. Lack of common vocabularies shared between computer (and pervasive) games for the purpose of applying MDSD (C1)
2. Lack of a consolidated definition of pervasive game (C2)
3. Lack of a structured domain analysis process to reuse domain knowledge efficiently (C3)
4. Lack of consideration of computer game traditions when applying the MDSD process (C4)

Firstly, although domain analysis plays an important role in the overall DSM approach, few scientific papers addressed how they performed the Domain Analysis (DA) process, as well as how the DSL concepts were made (as reviewed in section 2.6). As said by [18], common vocabularies are expected in order to automate game design (as an approach to regulate, formalize and reuse domain knowledge). However, not all vocabularies are qualified for DSM use. Concepts within such vocabularies should be of proper complexity, abstraction level and form so that it is feasible to implement them for the automation needs. Existing vocabularies (see section 2.1) do not meet these requirements. Thus the first challenge to this thesis was drawn up: lack of a common computer game vocabulary for the purpose of MDSD (C1). Further, such vocabularies should be able to present the knowledge of interests. As reviewed in section 2.1, there is no vocabulary describing pervasive games. In fact, there is no commonly agreed definition of pervasive games (C2). As the research focus of this thesis is about the concepts and modelling of pervasive games, these two challenges will first need to be solved. Later, how to apply such vocabularies in the domain analysis process in an effective and efficient way also deserves consideration (C3).
Secondly, as more research begins to adopt DSL and language workbenches as the MDSD approach for games development (see section 2.6), most have adopted DSM workflow directly without considering any of the peculiarities of computer game development (C4). As introduced in section 2.2, game software development should be connected with game design (and level design) tightly, and this development usually involves numerous prototypes. This is because gameplay often plays a decisive role in the success of the game, while whether or not the gameplay is interesting enough can hardly be decided before one can really play the game. As indicated in [117], a prudent and practical way to introduce new techniques into an existing production environment implies not only the new software, but also the new process and environment which must be integrated into their legacy equivalents. The process of MDGD must address the original procedural specialities of computer game software development before practitioners can use it and benefit from it.

3.2 Research Questions
In Chapter 2, the state-of-the-art for model driven game development was reviewed. The four challenges drawn from the state-of-the-art review can be grouped in two categories: conceptual (C1 and C2) and procedural (C3 and C4). Thus two research questions are accordingly proposed:

- **RQ1**: What important concepts need to be considered regarding creating pervasive games with a model driven approach?
  
  This research question attempts to address C1 and C2. The goal is to first create a clear definition of what a pervasive game should be, then construct a more detailed concepts set (domain vocabulary) which can be used to enhance the domain analysis of model driven pervasive game development. The definition of the pervasive game should be based on existing literature and have a reasonable scope for the research of this thesis, by detailing all pervasive features. Further, the domain vocabulary should provide appropriate concepts to realize such pervasive game features, and the concepts within the vocabulary should be of proper complexity and abstraction level so that it is feasible to realize them in DSLs. Two sub questions are accordingly raised:
  
  o **RQ1.1**: What important characteristics should/ may a pervasive game have?
  
  o **RQ1.2**: What concepts can be used in a Domain Specific Language (DSL) of pervasive games?

- **RQ2**: How can MDSD techniques be applied in a traditional pervasive /computer game creation process?

  RQ2 is used to address the procedure issues for both the DA part (C3) and the overall part (C4). The overall process should respect computer game development traditions (tightly connected game design and game software development, and the highly iterative process). The traditional DSM process must be adapted to match the overall process requirements while being kept efficient. As an important part of the overall process, how to use the domain vocabulary to perform DA in an efficient way should also be regulated. The following two sub questions are derived:

  o **RQ2.1**: How can a formalized domain vocabulary be used to enhance the domain analysis process in order to create pervasive games with a DSM approach?
RQ2.2: How can a traditional computer game development process be adapted to support DSM tasks in an efficient and iterative way?

3.3 Research Methods

Information systems (IS) are implemented within one organization in order to improve the effectiveness and efficiency of that organization. There are two paradigms in IS research: behavioural science and design science [23]. As behavioural science develops and verifies theories that explain or predict human behaviour or organization, design science tries to improve or extend the capacities of man or organization by building new artefacts [23]. The authors indicated that “in the design science paradigm knowledge and understanding of a problem domain and its solution is achieved in the building and application of the designed artefact”. Also, the goal of behavioural science is truth, while the goal of design science is utility. The research of this thesis is about applying model driven techniques in the pervasive game domain, and aims to create as its main contributions artefacts like conceptual model, vocabulary or engineering methods. Thus this research should follow the design science paradigm. [23, 176] are referred to as the guides used to develop the main methodology to define the research contributions and activities, as well as the overall process. In those papers, seven guidelines (see Table 2) were proposed and have been used very widely for understanding, executing, and evaluating design science research. These guidelines will be used in Chapter 8 to evaluate the research.

Design is both a process (activities) and a product (artefacts). In [25], the authors identified four design artefacts that can be produced by design-science in IS. These artefacts are constructs, models, methods and instantiations. Accordingly, this thesis defines three contributions (which will be introduced in detail in Chapters 4-6) according to the research questions proposed in section 3.2. Among them, RC1 (the TeMPS framework) and RC2 (the PerGO ontology) can be thought of as a combination of constructs and models, while RC3 (the GCCT workflow) is a method. As defined by [25], constructs or concepts come from the domain vocabulary, and they “constitute a conceptualization used to describe problems within the domain and to specify their solutions”, model is “a set of propositions or statements expressing relationships among constructs”, and method is “a set of steps (an algorithm or guideline) used to perform a task”.

- RC1 (Answering RQ1.1): A conceptual framework named TeMPS (meaning Temporality, Mobility, Perceptibility and Sociality) to summarize important characteristics of pervasive and social games.
- RC2 (Answering RQ1.2 and RQ2.1): An ontology named PerGO (meaning Pervasive Game Ontology) to structure and accelerate domain analysis for model driven pervasive games development.
- RC3 (Answering RQ2.1 and RQ2.2): A process named GCCT (meaning Game Creation with Customized Tools) to make use of model driven techniques within the traditional computer game development process.

Around these contributions, seven research activities were identified according to their importance and the time needed to investigate them. These activities are about building or evaluating the different contributions discussed above. Build and evaluate were the two design activities identified by [25]. Building is the process of constructing an artefact for a specific purpose, and evaluating is the process of
determining how well the artefact performs. [176] identified six activities based on the seven guidelines proposed by [23], as Table 10 presents. Among them, the problem identification, objective for the solution, and design and development are considered to focus mainly on constructing an artefact (RA1, RA3, RA5), while demonstration, evaluation, and communication are more about evaluating artefacts (RA2, RA4, RA6 and RA7).

- RA1: Construct the TeMPS framework by reading literature (regarding various pervasive game cases and theories), analysing common characteristics, and creating synthesis.
- RA2: Validate TeMPS by reviewing pervasive game literature and fitting corresponding pervasive games into the framework.
- RA3: Construct PerGO by reviewing literature (regarding computer game conceptual frameworks and computer/pervasive game design), refining and deriving (from TeMPS) concepts, and formalizing these concepts in UML diagrams.
- RA4: Evaluate PerGO by performing domain analysis for an example game and discussing this according to the SEQUAL framework.
- RA5: Construct GCCT workflow by reviewing literature (MDSD, computer game development, and MDGD) and practicing MDSD techniques on the Eclipse platform [27].
- RA6: Evaluate the effectiveness and efficiency of the GCCT workflow by constructing a pervasive game DSL (RealCoins) based on MDSD and PerGO, discussing subjective experiences, and analysing working hours and code lines.
- RA7: Evaluate the degree of user acceptance for PerGO and GCCT by performing a survey based on TAM (including constructing another small game DSL, e.g. RealPacman).

<table>
<thead>
<tr>
<th>Research Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Problem Identification and Motivation</td>
<td>Define the specific research problem and justify the value of a solution</td>
</tr>
<tr>
<td>2 Define the objectives for solution</td>
<td>Infer the objectives of a solution from the problem definition and knowledge of what is possible and feasible</td>
</tr>
<tr>
<td>3 Design and Development</td>
<td>Create the artefact (potential constructs, models, methods, or instantiations)</td>
</tr>
<tr>
<td>4 Demonstration</td>
<td>Demonstrate the use of the artefact to solve one or more instances of the research problem</td>
</tr>
<tr>
<td>5 Evaluation</td>
<td>Observe and measure how well the artefact supports a solution to the problem</td>
</tr>
<tr>
<td>6 Communication</td>
<td>Communicate the problem and its importance, the artefact, its utility and novelty, the rigor of its design, and its effectiveness to researchers and other relevant audiences (such as practicing professionals) when appropriate.</td>
</tr>
</tbody>
</table>

To perform research, various research methods (for both constructing artefacts and evaluating them) can be used. Such methods can be classified in various ways (e.g., Objective versus Subjective [177], Inductive versus Deductive [178], and Qualitative versus Quantitative [179]). More detailed methods include Literature Analysis,
Survey, Experiment, and Case Study[180]. The choice of research method to be used is made depending on the artefacts to be made, the state-of-the-art situation, intended audience, etc. As indicated in [181], using multiple research methods allows for greater confidence in the research findings. The main methods used in this thesis are introduced below:

- Frameworks and Conceptual Model [182]
- Literature Analysis[182] / Library Research [182]
- Literature Search [183]
- Exemplar [184]
- Experiment [180] /Assertion [183]
- Case Study [180, 183]
- Survey [182] [180]

In [182], frameworks and conceptual models were identified as one of the fourteen research methods commonly used in information systems research (see Table 11). The method is defined as “research that intends to develop a framework or a conceptual model”. This method applies to the construction of both TeMPS and PerGO.

Also in [182], library research “is based mainly on the review of existing literature” and literature analysis was identified and defined as “Research that critiques, analyses, and extends existing literature and attempts to build new groundwork, e.g., it includes meta analysis”. This applies to the construction of TeMPS and PerGO as well. For TeMPS, the characteristics of games that claimed to be “pervasive” in the literature was analysed, and a new framework was built to better organize these characteristics in order to better understand, classify or design pervasive features. While for PerGO, the aim has been to make qualified abstractions that capture the semantics in the computer game domain while meeting the requirements as DSL concept candidates. Thus literature for both the DSM domain and the computer game design domain was examined and analysed.

In contrast to literature analysis, a literature search is mainly an evaluation method, as defined in [183]. It “analyses results of papers and other documents to confirm an existing hypothesis or to improve the data collected in one project, with more similar data”. This was used to evolve and validate the TeMPS framework.

The use of exemplars is widely recognized as a technique for early evaluation of modelling approaches [184]. This method is used to evaluate PerGO (analysing a pervasive treasure hunting example) during the early stage of its development.

Experiments are “sometimes referred to as research-in-the-small since they are concerned with a limited scope and most often are run in a laboratory setting” [180]. In [183], the authors developed a taxonomy for software engineering experimentations for validating technology. The taxonomy describes twelve approaches that are grouped into three broad categories: historical methods (using existing data), observational methods (little control), and controlled methods. For instance, assertion is an ad-hoc validation method in the observational methods group. Developers execute their own assertion experiment to see if one proposed technology is better than other alternatives. In order to validate PerGO at an early stage, an experiment/assertion (developing RealCoins DSL) was performed.
Table 11. Methodologies in MIS Research [182]

<table>
<thead>
<tr>
<th></th>
<th>Methodology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Speculation/Commentary</td>
<td>Research that derives from thinly supported arguments or opinions with little or no empirical evidence.</td>
</tr>
<tr>
<td>2</td>
<td>Frameworks and Conceptual Model</td>
<td>Research that intends to develop a framework of a conceptual model.</td>
</tr>
<tr>
<td>3</td>
<td>Library Research</td>
<td>Research that is based mainly on the review of existing literature.</td>
</tr>
<tr>
<td>4</td>
<td>Literature Analysis</td>
<td>Research that critiques, analyses, and extends existing literature and attempts to build new groundwork, e.g., it includes meta analysis.</td>
</tr>
<tr>
<td>5</td>
<td>Case Study</td>
<td>Study of a single phenomenon (e.g., an application, a technology, a decision) in an organization over a logical time frame.</td>
</tr>
<tr>
<td>6</td>
<td>Survey</td>
<td>Research that uses predefined and structured questionnaires to capture data from individuals. Normally, the questionnaires are mailed. (now, fax and electronic means are also used).</td>
</tr>
<tr>
<td>7</td>
<td>Field Study</td>
<td>Study of single or multiple and related processes/ phenomena in single or multiple organizations.</td>
</tr>
<tr>
<td>8</td>
<td>Field Experiment</td>
<td>Research in organizational setting that manipulates and controls the various experimental variables and subjects.</td>
</tr>
<tr>
<td>9</td>
<td>Laboratory Experiment</td>
<td>Experiment research in a simulated laboratory environment that manipulates and controls the various experimental variables and subjects.</td>
</tr>
<tr>
<td>10</td>
<td>Mathematical Model</td>
<td>An analytical (e.g., formulaic, econometric or optimizational model) or a descriptive (e.g., simulation) model is developed for the phenomenon under study.</td>
</tr>
<tr>
<td>11</td>
<td>Qualitative Research</td>
<td>Qualitative research methods are designed to help understand people and the social and cultural contexts within which they live. These methods include ethnography, action research, case research, interpretive studies, and examination of documents and texts.</td>
</tr>
<tr>
<td>12</td>
<td>Interview</td>
<td>Research in which information is obtained by asking respondents questions directly. The questions may be loosely defined, and the responses may be open-ended.</td>
</tr>
<tr>
<td>13</td>
<td>Secondary Data</td>
<td>A study that utilizes existing organizational and business data, e.g., financial and accounting reports, archival data, published statistics.</td>
</tr>
<tr>
<td>14</td>
<td>Content Analysis</td>
<td>A method of analysis in which text (notes) are systematically examined by identifying and grouping themes and coding, classifying and developing categories.</td>
</tr>
</tbody>
</table>

A *case study* is an observational method and is “a study of a single phenomenon in an organization over a logical time frame” [182], “an experiment where a certain attribute is monitored and data are collected to measure that attribute, in order to investigate a specific goal for the project” [183]. However, to distinguish it from an experiment, it is “sometimes referred to as research-in-the-typical in the sense that it is normally conducted studying a real project and hence the situation is typical” [180]. Two DSLs (RealCoins and RealPacman) were implemented in order to evaluate the
effectiveness and efficiency of PerGO and the GCCT workflow as case studies in this thesis.

A survey is referred to by [185] as research-in-the-large (and past), because “it is possible to send a questionnaire to or interview a large number of people covering whatever target population we have” [180]. A survey was performed for this thesis in order to collect and evaluate the user acceptance for PerGO and GCCT based on the RealPacman Case.

A summary of the methods used for each RC is presented in Table 12.

<table>
<thead>
<tr>
<th>Research Methods</th>
<th>Used for</th>
<th>RCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frameworks and Conceptual Model</td>
<td>Construct</td>
<td>RC1 and RC2</td>
</tr>
<tr>
<td>Literature Analysis[182]</td>
<td>Construct</td>
<td>RC1 and RC2</td>
</tr>
<tr>
<td>Literature Search</td>
<td>Evaluate</td>
<td>RC1</td>
</tr>
<tr>
<td>Exemplar</td>
<td>Evaluate</td>
<td>RC2</td>
</tr>
<tr>
<td>Experiment</td>
<td>Evaluate</td>
<td>RC2</td>
</tr>
<tr>
<td>Case Study</td>
<td>Evaluate</td>
<td>RC2 and RC3</td>
</tr>
<tr>
<td>Survey</td>
<td>Evaluate</td>
<td>RC2 and RC3</td>
</tr>
</tbody>
</table>

3.4 Research Process

The research process of this thesis follows the design science methodology. Design science involves the rigorous process to construct artefacts and evaluate them. The process is inherently iterative [23]. [186] describes the nature of the design process as a Generate/Test Cycle (see Figure 65). [176] developed a more detailed process model (see Figure 66) based on this design process (and several other researchers’ theories). In the model, the overall process is iterative, but there are six research activities (see Table 10, the first three correspond to Construct/Generate and the latter three correspond to Evaluate/Test).

Further, [176] suggested that researchers may start with different activities according to the different nature of their research. The process is structured in a nominally sequential order, but in reality researchers may actually start at almost any step and go outward. Four kinds of approaches are enumerated, according to the different possible
research entry points: A problem-centered approach starts with activity one if the idea for the research is from observation of the problem or suggested from future research from other researchers. An objective-centred solution can start with activity two. This can be adopted when developing an artefact, and can address an industry or research need. A design and development-centred approach can start with activity three, resulting from the existence of an artefact which has not been formally thought of as a solution for the explicit problem domain. Finally, a client/context initiated solution starts with activity four. It can result in a DS solution if researchers work backwards to apply the process retroactively. This could be the by-product of a consulting experience.

Figure 66. Design Science Research Methodology (DSRM) Process Model [176]

Figure 67. Information Systems Research Framework [23]

A problem-centered approach was selected for this thesis, and this approach followed the basic sequence of the nominal process. Challenges were identified, and research questions proposed (see sections 3.1 and 3.2). It was inferred what an artefact would
accomplish (e.g. a conceptual framework to capture important characteristics that pervasive games should/may have, see the summary of the research contributions in Chapter 1). The artefacts were designed and developed (see Chapters 4-6). Then TeMPS, PerGO and GCCT were demonstrated (TeMPS was used to review pervasive game characteristics, PerGO to analyse a pervasive treasure hunting game case, and GCCT to create the sample pervasive games RealCoins and RealPacman). The effectiveness and efficiency of the study’s approach was evaluated (see Chapters 4-6), and communicated to other researchers via scientific papers. A larger audience was also engaged via a survey regarding user acceptance of PerGO and GCCT (see Chapter 7), in order to receive feedback from e.g. practicing professionals.

However, the three research contributions were not constructed and evaluated separately. It is indicated in the information systems research framework (see Figure 67) proposed by [23] that IS research should solve the business needs from the environment (problem space), by applying existing foundations and methodologies provided by the knowledge base. In the research of this thesis, the TeMPS framework provided a conceptual base for the evaluation of PerGO, and PerGO provided a conceptual and procedural base for GCCT. Thus, overall, these artefacts were developed in an ordered way (from TeMPS to PerGO and through to GCCT). However, since PerGO and GCCT are more related to each other (GCCT uses PerGO, while GCCT also contributes to the evaluation of PerGO), the actual research process of PerGO and GCCT overlaps a lot (but GCCT starts later).
Chapter 4 TeMPS: A Conceptual Framework for Pervasive Games

This chapter introduces the conceptual framework TeMPS for characterizing pervasive games. This framework was used in order to make the research subject clear by answering the question “what makes a game pervasive”.

The motivation of this research is outlined, and the framework formalized. Next a review of about thirty pervasive games is presented, based on the framework. Related works are also discussed and summarized in this chapter.

4.1 Motivation

Pervasive and social games encompass an emerging game genre which brings more physical movement and social interactions to games. Players have shown an increased interest in such games [28]. As said in [20], according to the online Merriam-Webster and Oxford Advanced Learner's dictionaries, the word pervasive has a clear meaning. It is an adjective indicating (the quality of) any given object or concept to spread, diffuse, or go through something. However, the actual definition of a pervasive game is very complex and ambiguous. Some research suggested that pervasiveness occurs in different perspectives and levels of intensity, and we need to be aware of the context and the perspectives, while letting go of “the notion of coining the ultimate definition of pervasive games [20]”. Thus instead of defining “what is a pervasive game”, a framework is proposed that includes the most important characteristics for a game to be pervasive, with the aim to answer the questions “why the game is pervasive” and “how pervasive the game is”. Although there is some related work, which will be discussed later, no similar detailed conceptual framework is known to exist yet.

4.2 The TeMPS Conceptual Framework

To develop the framework, the literature was reviewed on more than 30 pervasive and social games in different subgenres. It was identified that those games differ from other traditional computer games in four ways: ‘where’ the game is played, ‘when’ the game is played, ‘how’ the game is played, and ‘who’ the game is played by. These categories seemed to cover all the possibilities well, but were obviously not providing enough detail.

The detailed items that fell under these four categories were listed, and the games were analysed using them. Each pervasive and social game was checked to see: 1) whether these items were precise and detailed enough to differentiate the game from other computer games; 2) whether these items were precise enough to differentiate the game from other kinds of pervasive and social games. When either of the two tests failed, the reason why this occurred was analysed, the items revised, and the game in question reviewed again using new items. The current framework was achieved after
several iterations and a final arrangement, and this framework has worked well for all of the games reviewed with it up until now. Since these games are selected from different fields and sub-genres, it seems reasonable to say that the framework could work for a larger scope also.

The framework was named ‘TeMPS’ according to the four perspectives it encompasses:

- Temporality: addressing the game’s temporal property, i.e. whether the game is played in a fixed time/round or not (open beginning and/or open ended);
- Mobility: addressing the spatial property, i.e. whether the game could be played anywhere or whether it is fixed in one place;
- Perceptibility: addressing how the game is mixed with reality, e.g. does the game construct the appearance of the player proxy in the game by sensing the player’s real world appearance? Does the player need to physically move to move virtually in the game?; And finally,
- Sociality: addressing the player’s relationship with and social influence of the game.

In [21], it is stated that “Pervasive games are no longer confined to the virtual domain of the computer, but integrate the physical and social aspects of the real world”. The four perspectives proposed in this thesis collaborate together to maximize this integration of the real world with the game world: Perceptibility enables different means to integrate the physical aspects of the real world – players can physically play the game (jump, run, speak, etc.) as they would play in the real world, and sense the game world (see, hear, touch, etc.) in a similar way to how they feel the physical world. Sociality integrates social aspects of the real world – players can play with real human beings in the same way that they would interact with them in the real world. Mobility and temporality extend this integration in both spatial and temporal dimensions – everywhere and all the time.

Each of the perspectives includes one or more aspects. These will be presented each in turn, and exemplified with selected games. How all of these selected games fit into the TeMPS framework is presented in the next section.

4.2.1 Temporality

Two options are proposed here: Fixed time/rounds, and Open ended, meaning the player could play or exit the game at any time. Fixed time and fixed rounds are combined because, from the temporal perspective, they provide a similar experience. Commonly, the player focuses on a clear goal and has a strong desire to achieve it before the game ends. During this time, the player is immersed and all that he/she does is gaming (the player does not take part in ordinary life). This is common for competitive games such as *Brainball* [187], *ARQuake* [188], etc.

On the other hand, if a game is open ended, the rhythm slows down, and often, the game is interwoven with everyday life or even becomes part of everyday life – for example, a player may play *Ere Be Dragons* while beginning a walking journey [189]. In such cases, you can hardly distinguish the virtual world from the real world (as the action of walking is both a part of the player’s everyday life and part of the game in-
Besides, in this kind of game, the player’s emotions are often naturally (directly or indirectly) introduced into the game, which further blurs the real-virtual borderline. *SupaFly* [98] is an example of a game which is more like a community-based virtual soap opera.

Although some games are designed to be open-ended, due to technical or economic reasons, there is ultimately an ending. This issue will be neglected when the games are reviewed later in this chapter.

### 4.2.2 Mobility

Three options are available for the Mobility characteristics:

- Games are played *fixed* in one place as most traditional computer games are;
- Games can be played in large-scale outdoor places *anywhere* (often also played in everyday life); or
- Games can be played where the player must *move* in one place and physical actions are needed to change gesture, posture, etc. due to the requirements of gameplay.

Smart toy ([Zowie Playsets](21)) and Tabletop games ([Story Toy](21)) are mostly played in a fixed place. The player is requested to move in all location-based games like *Can You See Me Now* [190] and *Uncle Roy All Around You* [21]. Further, players may also need to move due to indirect requirements, such as to collect some objects (*Camelot* [191] and *Treasure* [192]), or to snapshot somebody (*DARE!* [193]). Movement in one place usually is because the player’s posture, motion, etc. is sensed and used in the game. *QuaGauntlet* [21] is a good example of a game in which the players need to change their postures to aim, fire, and activate their shields.

### 4.2.3 Perceptibility

Perceptibility can be discussed in two ways: 1) how the game perceives reality, and 2) how the reality (perceived reality of the player) is influenced by the game, or how the virtual world is perceived (by the player) physically. There are several ways for computers to know the player’s status (motion, location, emotion, etc.) or the current context (weather, what is seen, what is heard, objects/people nearby) by various sensors (including commonly used camera and microphone) or other means. However, a limitation of current technologies is that while physical objects may affect virtual objects, it is more difficult to affect physical objects by virtual ones [21].

Apart from traditional methods that provide visual and audio effects, there are some other methods adopted by modern games which give force/tactile feedback or even move objects. The utilization of these techniques commonly makes games easier to learn, and more engaging.

Some of the currently available aspects from these two ways to discuss perceptibility will be presented later. These are termed aspects instead of options because they are not exclusive to each other. Instead, they can be used simultaneously in any one particular game. It is noticeable that there is no certain relationship between a device and what is used in the game through this device. For instance, a camera is commonly
used to capture sight, but with advanced computing techniques it can also be used to capture motion. On the other hand, to detect motion, you can either use a camera or an acceleration detector.

1) How games perceive reality
Unlike generic input devices which lock people’s hands and arms in relatively similar hand postures and spatial location, pervasive games often utilize advanced input devices which enable and encourage players to play in a more natural way.

Sight: What the player sees, as in BREAKOUT FOR TWO [194], distant players see each other by camera and a projector. Sight is more widely used in Augmented Reality Games through head-wearable cameras.

Gaze: Indicates where a player points or gazes at. This is typically used in a game to shoot at somebody or change camera orientation, etc. (Swordplay [195]).

Speech, Sound: Use a microphone to capture sound/voice around/from the player. These inputs should be used as direct input to the game, not purely as a communication channel between players. In Tabletop WarcraftIII [196], meaningful speech is mapped to command primitives provided by the system, e.g. saying “unit one” and “build farm” will result in a command being issued to unit one to build a farm.

Emotion/ Heart Rate/ Physiological signal: This is a fascinating way for the game to perceive input from the user. The player could play by simply thinking hard (Brainball [187]) or trying best to relax (Relax to Win [197]), or could cause the game proxy to actually laugh by doing the same (AffQuake [21]). As a further example, Ere Be Dragons [189] is played when the player walks to work. The game detects the player’s heart rate and creates artistic effects according to it.

(Hand/Body) Gesture/Posture, Motion, Balance, Acceleration of Motion: Sense and use e.g. the player’s gesture, posture (AquaGauntlet [21], Control Freaks [198]), motion/acceleration (Swordplay [195]), balance status (Tilt-Pad Pacman [199]) to enable the game proxy to act accordingly. Motion and acceleration are sometimes hard to distinguish between, because they often occur at the same time. This is one of the reasons why motion is often detected/ calculated by an accelerometer sensor /accelerometer in addition.

Location/Route: The player’s (or an object’s) physical location is used to locate the game proxy on the virtual map. This information could be further used to produce correspondingly meaningful artefacts or images in the virtual world. For instance, in Ere Be Dragons [189], a hill is presented when the player passes a bus stop. Commonly, position information is obtained by means of GPS. However, in some games, location is reported by street gamers to online gamers (manually changing the agent’s position on a virtual street) using a mobile phone. In this thesis, this game is still regarded as location-based because the game rules require that the gamer physically moves to the place before he/she can report. Although commonly pervasive games sense the player’s context automatically instead of manually, this exception could be due to technical constraints or implementation challenges (to save money for instance).
Route is a variant of Location, meaning that the route the player/object moves along is used in the game by continuously sensing the physical location and calculating afterwards. *Your way Your Missions* [200] provides an example of this.

Object sharing: It is tricky to share an object between a game and reality. A common sharing technique is the RFID technique, while an easier technique is e.g. the one used in *Uncle Roy All Around You* [201], in which online players share some information (e.g. “to find a man in red” or “to find the man on the street corner”) with street players, therefore involving objects or persons in the physical world in an indirect way. Compared with the coins in *Treasure* [14], which serve purely as shared objects, a racket in *PingPongPlus* [21] serves as a detector at the same time. *Mona Lisa Bookshelf* [202] is a very special case of an object shared between reality and the game. In this application, a physically existing bookshelf is monitored and a virtual effect is presented according to the bookshelf’s status. What is special is that the shared bookshelf is the only medium through which the application receives input, and the application does not take into account who changes the bookshelf status or when and how the status is changed. It is noticeable that the shared object differs with the mediated sight. In the mediated sight, although physical objects exist in the game world also, they are treated purely as images instead of individual objects. Thus, we do not regard these physical objects as shared.

2) How reality is influenced by the game

Sometimes, reality is not influenced by the game at all. This is the case in e.g. *Camelot* [191] where an independent virtual world does not exist, and the computing technique is purely used as a traditional game utility. However, this is admittedly debatable since it could be said that the virtual game overlaps with the physical game, and that the two share the common game entities as well as the common game play and etc. In most cases even if it is difficult to distinguish the game and the reality, it is easier to identify how the reality is influenced.

a) Visual Effect: Show video, image, text which players can sense with their eyes.

The impact on the player is highly decided by the display device, for example head-mounted displays (HMD) (as used in *Human Pacman* [203]) can possibly have a more shocking impact because it is all that the player can see. A surround screen VR system (*Swordplay* [195]) presents things in a more natural way than a small 2D display on a device like a PDA. Visual Effects are usually accompanied by Auditory Effects. However, there are exceptions, as in *Virtual Aquarium* and *Mona Lisa Bookshelf* [202].

b) Blended Visual Effect: Blended Visual Effects are commonly used in AR games in which the gamers see the mixture of actual vision and virtual objects. Head-mounted displays together with a camera and hand-held devices are the most commonly used techniques to display such blended visual effects.

c) Auditory Effect: This is related to background music, sound effect, etc. Similar to the Visual Effect, the impressions differ with the use of different devices. Further, pure auditory effects (without Visual Effect) are often utilized in low-cost game environments like *Story Toy* [204].
d) Tactile/Force Effect: This is typically a force feedback as commonly used in console games. However, it is not found in the games reviewed in this chapter.

e) Move Objects and Others: This aspect needs to utilize some sort of smart machine to manipulate objects physically according to computer programs. Only one of the games reviewed, *Brainball* [187], has this feature. There are other ways for a game to affect reality. For instance, although many information systems like online exam systems are not physical, they are part of the real world. When a game affects such information systems, reality is affected. This was not found in the games reviewed in this thesis, however.

### 4.2.4. Sociality

Sociality is reflected in two aspects.

1) **Player Relationship**

Addresses the relationship between players, or what kind of social relationship can be felt between the players. Four options are proposed. 1) *Individually* – the player does not feel the existence of others in the virtual world when he/she plays the game. 2) A *Simple Collaborative/Competitive/Opposed relationship* is widely used in short duration games, bringing a clear but comparatively simple social air: friendly or hostile, etc. 3) The *Combined* option (Team Competitive like *Mobile Threat* [100]) uses several of the above options and makes the relationship between players a bit more complicated. 4) The *Community* option is the most natural way to experience the existence of other players and interact with them. In games such as *DARE!* [193], players are commonly allowed to change their roles and relationships with each other in an easy and flexible way. A free relationship further extends flexibility – players can change their roles even to be part of the audience or non-players, or vice versa [91]. An Open ended game may be an example of a free relationship, because it often stays dormant for a long time (no active player), but alerts the player to continue playing it at a given moment, or comes to life when somebody (who becomes the player) awakens it in some way. Player relationships often reflect the game’s scalability. Few games involve simple relationships among hundreds of players, and conversely, few virtual communities are built with less than 10 players.

2) **Social Influence**

Apart from the factor of pure entertainment, which is provided by most traditional games, more applications use interesting game elements but also have other goals and aim to have more social influence. Examples of the latter include education/learning/training (*Camelot* [191] trains children to collaborate), health improvement (*Ere Be Dragons* [189] monitors players’ health status while they walk), and personal management (*Virtual Aquarium* [202] helps players to build good tooth-brushing habits).

### 4.3 A Pervasive and Social Games Review by Utilizing TeMPS

The pervasive games were selected in different subgenres according to [21], and reviewed by characterizing them according to the TeMPS framework:

- “Smart Toys”: (*Zowie Playsets* [205], *Story Toy* [204]),
- “Affective Gaming”: (*Brainball* [187], *AffQuake* [21], *Relax to Win* [197]),


- “Augmented Tabletop Games”: (STARTS [21], Treasure [192]),
- “Location-based Games”: (Can You See Me Now [206], Uncle Roy All Around You [201], Human Pacman [203], The Drop [96], Capture the Flag [97], Epidemic Menace [99], SupaFly [98], The Songs of North [207], Insectopia [208], PAC LAN [209]),
- “Augmented Reality Games”: (AR² Hockey [21], PingPongPlus [210], Aqua-Gauntlet [21], ARQuake [188], ARTankwar [21], Magic Land [211], Tilt-Pad, Pacman [199], etc.), and
- Others: like Swordplay [195], NEAT-o-Race [101], Virtual Aquarium [202], Mona Lisa Bookshelf [202], Swordplay [212], Human Trials [213], and Control Freaks [198].

Apart from these kinds of games, several applications are also reviewed in this thesis which may hardly be called computer games, like Camelot [191] and Ere Be Dragons [189]. Since these applications have some typical characteristics, reviewing them should be helpful to both understand the framework and explore game design hints.

Since pervasive games can be classified as an emerging game genre which usually utilizes some high-tech devices or services, there is a large possibility that the games are constrained by technical factors when deployed (for instance, location-aware games may not work well in some places where the GPS signal used is not available). For the purpose of this thesis, the main consideration will be what a game is designed to be instead of the actual working implementation.

When reviewing the games, instead of assigning a simple classification to different categories, a score was proposed for each game. The games that had higher scores were regarded to be somewhat more “pervasive” because they utilize more ways to make the player feel “immersed”. Although this model of evaluating “pervasiveness” is not very accurate, it gives some strong indications. The following criteria are used to assign the score:

- For each game, all the aspects of Temporality, Mobility, Perceptibility and Sociality are considered and scored individually, and the sum is given as the final score of the game;
- For Temporality, a score of 1 is given to the game if it is Open ended, 0 is otherwise given;
- For Mobility, a score of 1 is given if the game could be played Anywhere, 0 is given if the game could be played in a Fixed place, 0.5 is given if the game could be played in one place but required the player to Move physically;
- For Perceptibility, which is considered to be more influential than other perspectives (although a bit subjective), a total score of 2 is shared by all the aspects involved, and 1 is assigned to the aspects in each category. Thus, for each aspect related to how the game perceives reality, a score of 0.1 is given if the game utilizes a corresponding technique; For each aspect related to how reality is influenced by the game, a score of 0.25 is given if the game utilizes a corresponding technique;
- For Sociality (Player relationship aspect), a score of 1 is given if the game allows a free or community relationship between players, 0 is given if the game is played by individuals only, 0.3 is given if the game allows simple player relationships (e.g. collaborative, competitive), 0.6 is given if the game allows a more complex player relationship (e.g. group competitive);
For the Social Influence aspect, a score of 1 is given if the game provides a non-entertainment influence, 0 is given if the game is purely for entertainment.

Figure 68 shows how all of the games reviewed were characterized using the TeMPS framework and scored. The following information can be deduced from this figure:

- By comparing the final score of each game in the first column, we can see which games are more pervasive according to the scoring criteria (commonly they provide pervasiveness through considering additional means);
- By looking at the scores in one aspect related column, we can pick out games that exemplify that aspect and the corresponding techniques (e.g. Tabletop WarcraftIII [196] is the only example for the Speech aspect);
- By looking across one row at how one game is scored for different aspects, we can see what kind of “pervasiveness” the game provides;
- By comparing two games in two rows, we see which game is more pervasive and what different kinds of pervasiveness they provide individually (e.g. Brainball [214] and DARE! [193] are pervasive in quite different ways).
- There are more interesting facts that can be derived from the figure. For instance, although the aspects in perceptibility are independent of each other, few games utilize more than 3 of them. Are the game designers simply not aware of this or do they feel that it is difficult to design attractive game play, or are they blocked by other practical issues? Considering the reasons may
lead to formulating guidelines to design a new pervasive game, or make an existing game more pervasive. Further findings include:

- Speech is seldom used;
- Location is widely used while Route is not;
- Motion is widely used while Balance is seldom used;
- Unlike most AR games, BREAKOUT FOR TWO [194] is creative in using Sight;
- STARTS [21] uses a smart way to blend the virtual world and the physical world, compared to most other games;
- Object sharing could not be introduced in a large scope because most techniques available (e.g. blue tooth, Wi-Fi) either need a power supply or are limited in their functions (e.g. RFID only provides static information, such as what it is). We may need to be ready for when these technical obstacles are overcome;
- Most games provide a Visual Effect and an Auditory Effect at the same time, but there are some exceptions;
- Few of the games reviewed provided a Force Feedback or Moving Object although it seems effective to use such a feature to bring “pervasiveness”.

4.4 Discussion

Although there is some related work in this area, no detailed conceptual frameworks similar to TeMPS have been found.

Nieuwdorp proposes a model [6] that describes pervasive games according to two perspectives: a technological perspective that focuses on computing technology as a tool to enable the game to come into being, and a cultural perspective that focuses on the game itself and how the game world can be related to the everyday world. There is very little overlap with Nieuwdorp’s model and the TeMPS framework.

Walther’s model [8] describes pervasive games according to four axes: 1) distribution – pervasive games communicate over various networks and are considered to be always online; 2) mobility – pervasive games give new challenges in terms of computing mobility, network mobility, user mobility, and context-aware services; 3) persistence – pervasive games demand total availability all the time; and 4) transmediality – pervasive games use various kinds of media sent back and forth over the network. However, some conceptual parts and technical parts are mixed. The TeMPS framework focuses on the concept level independent of technical implementation, although some game play is heavily restricted by the technical part.

The claim is not made that the suggested set of perspectives or their options/aspects that are used in this chapter are exhaustive, and they can be refined later. For instance, Sociality plays an important role, and another important aspect besides the ones mentioned here is how the players perceive each other’s existence and how they communicate (face-to-face real-time video and audio, or pure text mails?) [215]. Analysis in this area could be inputted to extend the TeMPS framework in the future. It is also realized that this framework is somewhat empirical and lacks a solid theoretical ground, which should be improved. In addition to these issues, the scoring criteria need to be improved in order to be less subjective.
4.5 Summary

In this chapter, TeMPS was introduced as a conceptual framework for characterizing the features that pervasive games may have. As indicated in Chapter 2, researchers have used the definition of pervasive game in different ways, and no commonly agreed definition is available. TeMPS provides an approach to better understand what are pervasive games and what features we should consider when trying to develop such games. TeMPS serves as a base for the other research contributions of this thesis, like PerGO (which will be introduced in Chapter 5). TeMPS was evaluated by a review of approximately thirty pervasive games, where games were fitted into TeMPS and their pervasiveness scored.
Chapter 5 A Pervasive Game Ontology (PerGO) towards Model Driven Game Development

In this chapter, the Pervasive Game Ontology (PerGO) is introduced. PerGO was developed based on an understanding of pervasive games (through TeMPS, as introduced in Chapter 4) and general requirements from model driven techniques. PerGO was used as the basis of the domain analysis part in the GCCT workflow (which will be introduced in Chapter 6).

Firstly, the domain analysis approaches are introduced, and the motivation of this research is outlined. Then the formalism of PerGO is presented, and a domain analysis procedure based on PerGO is proposed. Later, the usage of PerGO is demonstrated by an example. Lastly, the ontology is evaluated and this chapter summarized. The evaluation of PerGO consists of two parts: one is subjective/analytical (which is presented in this chapter), and the other is more objective (which will be presented in Chapter 7).

5.1 Background

As mentioned in Chapter 2, using models has been a long-standing tradition to design complex software systems, and it has become even more popular when the Unified Modelling Language (UML) was developed [216]. Models help us to understand a complex problem and its possible solutions. However, they have been perceived primarily as documentation artefacts, and thus the creation and usage of them has seemed to be peripheral to software development [108]. This situation changed by the application of MDSD, where models became the primary focus and products of software development rather than computer programs [117].

The motivation of MDSD is to move the working focus from programming to solution modelling [118]. This is achieved by two important mechanisms: providing abstractions that are closer to the problem domain and generating programs from their corresponding models [118]. By doing so, a number of potential benefits can be gained. Firstly, models/software are easier to understand and specify. It is even possible that domain experts or other people without a programming background can write full specifications and produce systems [117]. Secondly, models are easier to maintain than code. Using concepts that are closer to the problem domain makes models less sensitive to the chosen technology of their solutions and to the evolutionary change brought by the technology [117]. Thirdly, the software development productivity can be significantly improved [217]. The common application fragments are written and encapsulated in templates, then used for many models to generate codes. And lastly, the quality of software can be improved. Polishing this common part of code templates allows all models to benefit from it.
To play to its strength, MDSD should be **domain specific** [17]. This requires domain specific languages (instead of General Purpose Languages) to raise the level of abstraction, and domain specific platforms to reduce the complexity of the code generator. That is why Domain Specific Modelling (DSM) is regarded as an alternative terminology to MDSD in many scenarios. These two terminologies are used interchangeably in this thesis.

As DSM has been applied in many domains and successfully achieved expected benefits, researchers have tried to apply it to other domains such as the computer games domain [10-15]. The computer games domain seems to be desirable for this application due to its complexity and traditions. Computer games can be extremely complex due to complicated architecture and different domain knowledge. Applying DSM can potentially alleviate the overall complexity by separating and hiding the domain (knowledge) complexity in Domain Specific Languages (DSLs). Further, it is quite common for game software to utilize a generic game engine which executes specific level descriptions that are produced in a level editor. DSM proposes a similar approach: utilizing generic code patterns which execute specific data (model) that is produced in a DSL editor, but in a more structural way. DSM applies as well to games that do not have a fully functional game engine or level editors. Also, since DSL editors can be automatically generated (with a meta-model as the only mandatory input) by utilizing language workbenches, the overall automation degree is expected to be higher.

However, the application of DSM in computer game domains is still not common and mature. Although DSM is able to provide convenience and benefits to software development (as stated above), the development of DSM is not easy. For instance, DSL, which is one of the most important artefacts in DSM solutions, has two major challenges: the abstraction challenge (how to provide support for creating and manipulating problem-level abstractions) and the formality challenge (how to formalize the semantics and what aspects of semantics need to be formalized) [108]. To meet these challenges, domain analysis plays an important role among all the four tasks (decision, analysis, design, and implementation [218]) to develop a DSL. This is because the outputs (usually including domain definition, domain specific vocabulary with semantic meanings, and a model describing commonality and variability space) from domain analysis support the decision making up front, and provide a solid base to start design and implementation afterwards. A structural and efficient domain analysis can be crucial to the quality of DSL and the success of the overall DSM solution.

However, few DSLs (only four out of 39 as evaluated in [218]) have utilized a more formal domain analysis such as FAST (Family-Oriented Abstractions, Specification, and Translation) [219] and FODA (Feature Oriented Domain Analysis) [112]. The formal domain analysis has shown good language design results, but the usage of these approaches is still limited [220, 221]. Researchers have investigated the Ontology Based Domain Analysis (OBDA) [220] since ontologies are closer than their original domains and provide an effective way to reuse the domain knowledge in traditional domain engineering [222]. This stems from the close link between domain analysis and knowledge engineering. Knowledge capture, knowledge representation, and ontology development can therefore be useful [218]. Other research following this direction in various domains include [221, 223-226].
Among all the articles describing the various efforts towards Model Driven Game Development (MDGD), it is difficult to find comprehensive descriptions about how their domain analysis was carried out, or where the abstractions in their DSLs came from. The work in [227, 228] may be the only illustration of the detailed domain analysis process and the result structure. In their research, core dimensions for the game development were considered and analysed, and results were recorded in a feature model with almost 150 features (which describe the commonality and variability). Most other works came up with a set of DSL concepts directly, without saying how they were identified and why they were chosen (see Chapter 2 for more details).

Insufficient domain analysis may partly contribute towards unqualified DSLs and the unsuccessfulness of DSM applications, hence the inspiration for the research of this thesis. In this chapter, an ontology – the Pervasive Game Ontology (PerGO) – is proposed as an important means to aid domain analysis (similar to OBDA [220], introduced previously). In order to make the work more concrete, the focus is on pervasive (computer) games instead of general computer games. The ontology is used at a domain modelling level (not for process management or other purposes [229]). By using the ontology to structure and accelerate the production of the main domain analysis outputs (concepts, commonality and variability), it is expected that more qualified DSLs can be produced in an efficient way. Two benefits are expected: 1) to accelerate the domain analysis process by proposing a number of pre-made domain specific artefacts (concepts and relations) which can be used directly or with minor modification to generate domain analysis outputs; 2) to regulate and structure the domain analysis process by proposing ordered steps based on PerGO.

Figure 69 presents the expected progression from domain knowledge to DSL models. Firstly, an ontology (PerGO) is used to formalize the common domain knowledge for pervasive games (see section 5.2). Then DSL meta-models are constructed by customizing the ontology for the more specific domains like pervasive treasure hunting games (see section 5.4). When the DSL is ready to use, it will be possible to write specifications (DSL model) using the DSL. Such specifications are instances of the corresponding domain model (DSL meta-model) and can be used to generate game software by utilizing MDSD tools.

**Figure 69. From Domain Knowledge to DSL Model**

### 5.2 Pervasive Game Ontology (PerGO) Formalism

PerGO is designed to serve as a structured and efficient domain analysis. The motivation behind PerGO is to pre-define a number of commonly used concepts (as well as the relationships between them) within the pervasive game domain, and provide a structured way to use these concepts to produce domain analysis outputs, and construct DSM artefacts afterwards.
There are in total more than 100 concepts in PerGO, and they are organized into 6 perspectives which focus separately on different aspects of game software. These perspectives are Gameplay (challenges and actions), Artificial Intelligence (AI), Virtual Game World, Control, Presentation, and CtrlPresentation (an abstraction for logic which is in charge of flexible user interfaces such as physical user interfaces). In addition, there are two kinds of concepts within PerGO: high-level concepts which are common to all computer games and low-level concepts which are specific to pervasive games (primarily used or often used by pervasive games). While high-level concepts constitute the core part of PerGO, low-level concepts are derived from the high-level concepts and constitute the pervasive part of PerGO. This way of splitting the overall ontology into two parts is common. Researchers call these two parts “Upper-level Ontology” [230] or “Frame Ontology” and “Domain Ontology” [231], and usually concepts of the latter part are inherited from concepts of the former part [230, 232].

![Figure 70. Structure of PerGO](image)

Figure 70 shows a sketch of the PerGO structure. PerGO concepts are located in a two-dimensional (Perspective and abstraction level) space. The structured organization of PerGO brings two advantages: First, by going through an ordered combination of perspectives and abstraction layers, the process of domain analysis can be regulated in a structured way. Second, it is easier and more efficient for the domain analyser to use concepts or derive new concepts if there is no proper candidate.

5.2.1 Design Decisions

Six perspectives were defined based on useful sources to identify DSL concepts [17] and on the current research focus of this thesis. In [17], the authors suggested several useful sources to find language concepts, including the physical product structure, the look and feel of the system, variability space, domain (expert) concepts, and generation output. Among these sources, variability stands out since it can encompass all the others. The approach taken in this thesis proposes to pre-define pervasive game concepts in PerGO according to the other sources (domain concepts like gameplay and
AI, look and feel of the system like the presentation, and generation output like game elements), then customize the concepts according to the variability space of a more specific domain like pervasive treasure hunting games (see Figure 70). Reference is made to some frameworks in the computer game domain to make sure that the domain analysis (in the form of six perspectives) is consistent with game design traditions. The sequence is decided among perspectives, with a careful aim to better handle potential design dependencies.

There are numerous concepts within one domain. PerGO focuses on identifying those that are promising to work as DSL concepts (ones that support automatic code generation instead of only conceptual modelling). To ensure this, three criteria have been made to select or make concepts in PerGO: 1) the concepts should be of proper complexity. Simple attributes that can be expressed by one variable of primitive type are thought to be of improper complexity and are excluded; 2) the concepts should be of proper abstraction level. Those concepts that appear in the discussion of game design or level design, and that at the same time are possible to implement as a class or some other encapsulated data structure, are chosen. This is to balance between the expressiveness and the flexibility of the DSL. High-level concepts which are difficult to implement or low-level concepts which do not appear in the common discussion are considered to be of improper abstraction level; 3) the concepts should be constructive. In other words, the concepts should describe nouns, such as something or some rule, not adjectives, such as ‘interesting’ or ‘intuitive’.

There are a number of languages that can be used to express ontologies [233]. In [234, 235], the authors discussed the possibility and feasibility to use UML as an ontology language. A UML class diagram [236] was chosen to formalize PerGO as well. This was done primarily because PerGO will be further used to construct DSL metamodels which are often based on UML class diagrams also. Therefore, the transition between domain analysis and DSL definition can be smooth. What is more, since UML is well known to many software developers, compared to other ontology languages, the threshold of learning to use it can be lower (other ontology languages like OWL [237] may not be so widely known to software developers).

A UML class diagram contains all the standard hierarchical relations [109] (classification, aggregation, generalization and association) that are needed to describe relationships among concepts. In a UML class diagram, the main constituents are classes and their relationships. A class is a description of a concept, and is represented as a rectangle. A relationship between two classes is drawn as a line. A normal association relationship is drawn as a line with an arrow at the end, while other kinds of relationships are represented by different kinds of lines. For instance, lines with a triangle at the end indicate association relationships, lines with a black diamond at the end indicate aggregation relationships, and dotted lines indicate dependency relationships.
5.2.2 Perspectives and the Core Part of PerGO

In PerGO, six perspectives were defined from which useful concepts can be identified. These six perspectives, as well as the main core part concepts included in them, are listed as follows with a brief introduction. Other concepts within this part will be introduced further in the next section since they are used to derive more concepts in the pervasive part of PerGO.

- **The Gameplay** perspective includes all of the concepts that describe the challenges and actions from the players’ perspective. While [Challenge] describes what the players have to overcome, [Action] describes what the players are allowed to do to overcome these challenges [44].
- The **Artificial Intelligence (AI)** perspective includes concepts which describe rules from the game elements’ perspective, regarding how game elements react to players, or how game elements evolve when no interaction happens. An abstract concept [AI] is defined in the core part standing for such logic.
- While Gameplay and AI depict the dynamic characteristics of a game, **GameWorldElement** focuses on the static ones, and contains all the concepts representing game elements like [Map], [Character], and [Element].
- To make the game interactive, **Control** focuses on how various physical inputs are used to control the game status, by invoking actions or changing variables. The concept [Controller] is defined in this research as the core part referring to the logic which accepts physical inputs, and controls part of the game world (which implements the interface [Controllable]).
- The **Presentation** perspective focuses on how the game utilizes various output devices to make the game status explicit, and influence the physical world. Similar to the control perspective, the two concepts of [Presenter] and [Presentable] are defined in the core part, corresponding to the presentation logic and the objects that can be presented.
The CtrlPresentation perspective is defined due to the complexity of control and presentation. Some special artefacts also exist which control and/or present the game world in various special ways. While Control and Presentation are usually connected with one and only one I/O device, [CtrlPre] is more flexible and can be connected with several different devices (speaker, display, force feedback gamepad, or physical UI devices). CtrlPre can be connected to both Controllable and Presentable, but usually these two interfaces are implemented by one game element. This causes CtrlPre to differ with each combination of Controller and Presenter.

An overview of the core part concepts within these perspectives is presented in Figure 71. In this figure, not only the concepts, but also the relationships among them (as introduced above) are captured.

5.2.3 The Pervasive Part of PerGO

There are in total around 100 concepts in the pervasive part of PerGO. In the following sub-sections, the concepts will be briefly introduced according to the perspectives they are included in. In each section, an overview figure is presented which shows all the concepts and relationships within its corresponding perspective. In these figures, coloured concepts (core concepts) are within the core part (grey ones from the current perspective, and yellow ones from other perspectives), while the white concepts are in the pervasive part. All the concepts in the pervasive part are derived from core concepts. The focus will be on introducing the concepts that are primarily used by pervasive games. There are some other concepts which are often used by pervasive games, but also by other computer games. These will not be discussed in detail since they are fairly well known.

5.2.3.1 Gameplay

In this perspective, two kinds of concepts are included: Challenges, meaning the goals that players are required to fulfil in order to win a game; and Actions, meaning the atomic things that players are allowed to do in order to respond to the challenges. In some games, many challenges can be achieved by one kind of action. For instance, MoveToCha (move to a place), GetCloseCha (get close to a place or area), EnterCha (locate inside an area), and AvoidCha (avoid being close to a place or area) can all be achieved by the action Move. On the other hand, it is possible that one challenge is achieved by performing different actions. For instance, GetObjCha can be achieved by Move (by co-locating only), or by Move and Get (by co-locating and an explicit action to get). Game designers should design the combination of Challenges and Actions carefully to achieve unique and interesting game experiences.

For the challenges, they are classified according to whether they can be fulfilled by an individual player, or by collaborating with other players, or by competing with other players. These concepts are derived from CollaborateChallenge and CompeteChallenge accordingly. An overview of the concepts within this perspective is presented in Figure 72.
5.2.3.2 AI
The AI perspective (as shown in Figure 73) includes two kinds of concepts which derive from InteractionRule and EvolutionRule respectively. While interaction rules describe how game elements interact with each other, evolution rules describe how game elements behave by themselves. Collide may be the most commonly used interaction rule which describes under what conditions (the distance between two elements for instance), a collision happens. In computer games, some events are often invoked when a collision happens. On the other hand, rules describing how one element reproduces itself under specific conditions (Duplicate), rules describing how one element finds the path to other places (MoveTo), etc. are also included in this perspective. In pervasive games, it is common that the status of some virtual element may need to be updated regularly according to the data sensed/ captured in the real world. UpdatePhysicalInfo can be used for such elements. It has become more popular recently to allow players to update part of the game information to social network applications like Facebook [238], Twitter [239], etc. UpdateSocialInfo can be used in such circumstances.

5.2.3.3 Virtual Game World
In the virtual game world perspective (see Figure 74), it is normal to have a Map as well as some elements corresponding to the map like Point (a position), Route (a series of points) and Place (an area). These MapElements are usually identified by different ways according to how the map is represented. For instance, in a location-based game, each point may be identified by a pair of latitude and longitude values. While in a traditional 2D virtual game, points are identified by a pair of coordinates for hori-
zontal and vertical dimensions. In addition, there are some WorldElements which are located at specific points or move about on the map. Several commonly used kinds of elements can be identified, including: Character (with a figure of a human being), PlayerProxy (characters which are controlled by players), and Group (a group of PlayerProxies which collaborate to play the game. This kind of element commonly only exists for a short time), Community (a social group of PlayerProxies which exists for a long time), etc.

**Figure 73. Pervasive Game Concepts in the AI Perspective**

**Figure 74. Pervasive Game Concepts in the Virtual Game World Perspective**

### 5.2.3.4 Control

Traditional computer games use traditional input devices like mice and keyboards to control the game. Touch screen is often used as big screen mobile phones are becoming popular. Pervasive games may adopt more kinds of input devices to involve physical elements in the game world. According to the input data, two kinds of Controllers are identified in this perspective: HumanPhyController, which receives and uses data regarding players’ physical information; and EnvironmentController, regarding physical information of the real physical world environment.
Figure 75. Pervasive Game Concepts in the Control Perspective

Usually, HumanPhyController is connected with some kind of sensor/camera/microphone/gyroscope, and collects the player’s information to invoke actions or change game status. Typically such information is about location or physiological signals (e.g. heart rate, brain waves). HumanPhyController can include logic to transform data from the control device to more meaningful or more advanced formats. For example, while a GPS device can only provide location information, RouteCtrl (which is connected to the GPS device) may record previous data, calculate the route according to this data, and control the game according to the route. What is more, pervasive games also use sensors to capture information from the environment (such as temperature and wind direction) to control the game status. Such Controllers are captured as EnvironmentControllers. Figure 75 shows an overview of various Controllers.

5.2.3.5 Presentation

Concepts in the Presentation perspective are shown in Figure 76. From the figure, we can see that, besides common video (VisualPre) and audio presentation (AudioPre), pervasive games may also use physical approaches (PhyPre) or electronic approaches (ElePre) to present game status. Typically such approaches include producing force feedback (ForceFeedback), moving objects (MoveObject), capturing a picture/video/sound (CapturePicture/CaptureVideo/CaptureSound), sending a short message (SendSMS), making a call (MakeCall), sending an email (SendEmail) or pushing data to other social network applications (ShareOnSocialApp).

5.2.3.6 CtrlPresentation

In this perspective, all concepts regarding logic which is in charge of both control and presentation are included at the same time (see Figure 77). Besides the commonly used graphical user interface components (GUIElement), the main view showing the map (MapView) and AirView (which overlaps on the main view to show a simplified bird’s eye view), pervasive games often also utilize PhysicalUIManager (to interact
with physical UI devices), MixedView (mixing the real sight of the player and virtual elements).

![Figure 76. Pervasive Game Concepts in the Presentation Perspective](image)

![Figure 77. Pervasive Game Concepts in the CtrlPresentation Perspective](image)

**5.3 A Domain Analysis Procedure Based on PerGO**

A four steps (domain analysis) procedure is proposed in order to produce outputs of domain analysis based on PerGO. As introduced previously, three major outputs are usually produced by domain analysis: a domain vocabulary, commonality space, and variability space. Table 13 was used to analyse and collect such information.

The four steps proposed are:

1. Quickly identify perspectives that are related to the current domain, and record them in the first column;
2. Go through the perspectives in the first column. For each one, consider the corresponding common game design, then select useful concepts from PerGO or derive more specific concepts for the domain based on PerGO to represent this design. Record the concepts in the second column, in case there is more
detailed information, especially attributes corresponding to the concepts, and record them in the third column;
3. Similarly to step 2), go through the perspectives in the first column. This time, consider the variable game design that may be used in different game samples. Decide whether some other concepts (within PerGO or newly invented) are needed besides those that have been listed in the second column, and add them if this is the case;
4. Go through all of the concepts in the second column, and decide how to utilize attributes of them or relationships among them to support the variable game design. Then write them in the last column.

<table>
<thead>
<tr>
<th>Perspective</th>
<th>Concept Commonality Details</th>
<th>Variability Details</th>
</tr>
</thead>
</table>

The four steps help analysers to make considerations in a reasonable order: from general aspects (perspectives) to concrete aspects (concepts, attributes, and relationships), and from common aspects to variable aspects (variability often depends on commonality). Further, whenever the perspectives are gone through, they should be gone through according to the sequence presented in this chapter: from Gameplay to AI, VGW, Control, Presentation, and CtrlPresentation. The rationale behind this order will be explained in section 5.5.

5.4 A Pervasive Game Example

In this section, the pervasive game example “PervasiveTreasureHunting” is examined, and described below. Most of the features of this game example have been previously realized in several prototypes, and thus the description can work as a simplified game design document. How to carry out domain analysis with PerGO by following the steps introduced in section 5.3 is illustrated. The use of exemplars is widely recognized as a technique for the early evaluation of modelling approaches [184]. The game description is as follows:

PervasiveTreasureHunting is one location-aware variation of traditional treasure hunting games. The game allows several groups of players to physically move within the Trondheim downtown area and play the game. They try to get notes from pre-defined places, finish the tasks described in these notes, and compete with each other according to which group or which player has the most game points collected by getting notes or finishing tasks. A pre-defined route can be made covering the places, and in that case, players will have to follow the route to get the notes. Different tasks can include moving to someplace, taking a picture of something, taking some objects from somewhere, and exchanging objects with other group members. Players may not get the game points associated with a post or a task if they use more time than the pre-defined time limit to find the post or solve the task.

Mobile devices are needed to play the browser game. The game should present a map that is annotated with places as well as the positions of players, and other game status information (like the amount of time that has past, the amount of time left, and who is the winner). The client-side application should allow players to login with a unique group ID and player ID, and present information in different way accordingly. For instance, icons of players within the same group of the current player should be dis-
played with a different transparency than those icons of other group players. For the server side of the application, it is required to use a map with a different style (a dark one ideally) and all the player icons are highlighted with full transparency. The players play the game mainly by physically moving, which will cause the player proxies to move in the game world. Players use one hot key to read notes or solve tasks.

Table 14. Domain Analysis Table for the PervasiveTreasureHunting Game

<table>
<thead>
<tr>
<th>Perspective</th>
<th>Concept</th>
<th>Commonality Details (Attributes)</th>
<th>Variability Details (Attributes &amp; Relationships)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Game World</td>
<td>Game</td>
<td>Name, Introduction, TimePast, TimeLeft, Winner</td>
<td>MaxNumberOfGroup, MaxNumberOfPlayer, GroupWinnerOr-IndividualWinner</td>
</tr>
<tr>
<td></td>
<td>Group</td>
<td>ID, GamePoint</td>
<td>Relates to Player</td>
</tr>
<tr>
<td></td>
<td>PlayerProxy</td>
<td>ID, GamePoint</td>
<td>Relates to Group</td>
</tr>
<tr>
<td></td>
<td>Note</td>
<td></td>
<td>ID, Position, GamePoints, TimeLimit, relates to Task</td>
</tr>
<tr>
<td></td>
<td>Task</td>
<td>Name, Task-Description</td>
<td>ID, GamePoints, TimeLimit, relates to Note, relates to Challenge</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gameplay</td>
<td>TakePictureCha</td>
<td>PictureID (to judge whether the challenge is fulfilled)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TakePicture</td>
<td>Relations to KeyboardCtrl</td>
<td>(to invoke the action)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results of the domain analysis are presented in Table 14. Due to limited space, only a part of the results is presented here. The upper part of Table 14 shows the concepts in the Virtual Game World perspective. These concepts are usually easy to find if we pay more attention to the nouns in the game description (see the words highlighted in yellow above). Some of the concepts need to be renamed in order to align with PerGO, and to maintain consistency with other DSLs that are developed based on PerGO. For example, PlayerProxy is used (from PerGO) instead of Player (which is used in the description) as the concept name. It is a little tricky to decide whether an attribute should be specified in the commonality (Step2) or variability (Step4) section. The key is, if the attribute needs to be assigned when constructing the codes, then the attribute should be specified in the variability section. If the attribute needs to be assigned at run time, it should be specified in the commonality section (meaning all the designs need this attribute). For instance, Game Points of a Note should be defined before the codes are constructed (in variability). While Game Points of a PlayerProxy should be calculated at run time (in commonality). These should be considered carefully because the content in commonality may not contribute to the meta-model of the DSL.

The lower part of Table 14 shows a Challenge concept (TakePictureCha) and an Action concept (TakePicture). Candidates for these concepts can be found by observing
the verbs that are used to describe scenarios about how players play the game (see the underlined words in the game description). Besides textual description for the specific game, sometimes additional details for the commonality and variability need to be complemented according to the semantics and common knowledge. As the variability shows, a KeyboardCtrl should be related to the action. Consequently, when a key is pressed, the action will be invoked. On the other hand, a picture ID should be assigned to the challenge, so that when a picture is taken, the challenge can judge whether it fulfils predefined requirements (sharing some common objects of the known picture with the ID, for instance).

After the table is filled in, it is time to construct an initial meta-model for the DSL. All the nodes were created corresponding to the concepts found in the table. If some concepts did not have any variability details, corresponding nodes were not created since they were not going to be used to decide how to generate the codes. Abstraction nodes were then added from PerGO, which can work as a container to contain previous nodes, or as a parent to derive previous nodes. All the concepts were organized as an aggregation tree, as it was required by the MDSD code generator. Attributes and other associations were added as last indicated in the variability section details. Figure 78 shows the result of the meta-model corresponding to the previous domain analysis (attributes are not included due to the limited space). From the figure, we can see that most of the concepts are directly from PerGO (the coloured ones).

Figure 78. Initial Meta-Model for the Case Study

5.5 Discussion
As previously introduced, DA is an up-front task in the overall DSM approach, and its quality may be crucial to the quality of DSM artefacts, including DSLs. However this part has not received much attention in many domains, including the computer game domain. Inspired by this situation, a way of performing DA was proposed, based on a pre-defined vocabulary (PerGO) to structure and accelerate the DA process. In this section, the quality of this approach will be discussed, in the sense of 1) whether the
pre-defined vocabulary provides proper abstraction for the target domain (effectiveness of the vocabulary); and 2) whether the overall process is reasonably structured (efficiency of the process). A brief introduction will be given before the discussion section, as to how other game domain analyses and game vocabularies (that were presented in the literature reviewed) perform for the purpose of MDSD practices. The cost and limitations of this approach will also be discussed.

5.5.1 Review of Domain Analysis Methods in MDGD

Some scientific papers were reviewed in the MDGD domain to gain an understanding as to whether domain analysis was formally performed to make abstractions of DSLs, and in what manner it was performed. The main aspects are reviewed below:

1. Was any formal DA methodology (like OBDA [220]) adopted?
2. Was the DA process (or domain design since the processes were described in some papers as a whole) structured (with clarified framework or steps)?
3. Was there any pre-defined vocabulary that can be used directly in the DA process?
4. Was the pre-defined vocabulary full-spectrum instead of focusing on one or two specific aspects of the game?

<table>
<thead>
<tr>
<th>Literature</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal DA Methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured DA/DD</td>
<td>*</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-defined Vocabulary</td>
<td>*</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full-spectrum Vocabulary</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSL (Not GPL)</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

It is actually difficult to find many papers presenting DSL or DSM practices in the computer game domain. Therefore, the review included some unfinished work and some work using MDA and UML as well (such as [160] and [148], since they are comparatively more related). Most papers introduced DSLs or workflow without illustrating much about the DA part in particular. Table 15 summarizes the main findings. In the table, each literature column is used for a work which is introduced in one or several scientific papers. The “*” symbol is used to indicate that the work meets the criteria specified in the first column. And “s” is used to indicate that the work meets the criteria in a semi-finished way: using a semi-structured process or semi-formal vocabulary.

The review revealed that no paper claimed to have used any formal DA methodology. Many of the papers proposed DSL concepts directly [11, 13, 126, 175], while some other papers identified several dimensions which contained the DSL concepts. For instance, in [15] two dimensions (visible structure and gameplay related entities) were identified. In [241], structure, behaviour and control mapping were used to structure domain designs. While in [160], although domain analysis was not mentioned explicitly, the same authors identified six areas to address game design concerns including gameplay, graphic interface, audio design, artificial intelligence, game storytelling...
and level design. Among them, only gameplay was addressed in the DSL. These practices are thought of as semi-structured domain analysis/design.

[227, 228] may be the work which is closest to the approach used in this thesis. In [227], the authors proposed ten dimensions to define a product line including User interface, Game flow, AI, Sound/Music, etc. A more detailed ontology introduced in [228] was used to define the SharpLudus product line (as the main part of the 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. Exactly how the product line definition (domain analysis) contributed to the construction of DSL concepts based on the ontology remains a question. In [148], a conceptual framework was introduced which consisted of a three tier design architecture (flow, scenarios and objects) and components (screen components, GUI components, In-game components, etc.) for serious games. This conceptual framework structures the abstractions and relationships, but might not be able to work as a domain analysis structure in the common way. Also, the framework encompassed serious games instead of common computer games. Some concepts were enumerated in the framework, which, for the purpose of this thesis, can be thought of as a semi-formal vocabulary.

As a result, although using formal methodology and pre-defined domain knowledge (e.g. as a vocabulary) were thought useful, these were generally not applied in MDGD. In the approach in this research, the attempted remedy was to propose a structured DA process based on a pre-defined domain vocabulary.

### 5.5.2 Towards an Effective and Efficient Domain Analysis Method

As introduced and reviewed, the expectation is for a formal domain analysis method that can be used for model driven game development. The method would be based on a domain vocabulary in order to reuse complex domain knowledge. To ensure the method’s effectiveness and efficiency, the following criteria are proposed to evaluate such a method (and the domain vocabulary that the method is based on):

1. **(Effectiveness)** The vocabulary should support the implementation of the domain specific features;
2. **(Effectiveness)** The concepts within the vocabulary should be constructive and of proper abstraction level and complexity to support the implementation of DSM artefacts;
3. **(Efficiency)** The overall process should be reasonably ordered to solve the design dependencies/ constraints (for instance, variabilities usually are decided based on commonalities).

As will be reviewed a bit later, most of the MDGD practices did not present any domain analysis structure or steps. Only a few of them (Literature B, C and D in Table 15) identified dimensions to group their domain analysis/design, but they did not articulate how and why these dimensions should be ordered (did not meet Criteria 3). Literature A is the only one found which presented both ordered domain analysis steps [227] and a comparatively formal vocabulary [228] that can be used to present domain analysis outputs, and thus the closest work to the research of this thesis. However, how the vocabulary was used to generate the domain analysis was not illustrated in detail. And since the vocabulary was designed to define a domain (probably not in the form of defining a DSL for the domain), it did not meet Criteria 2 (many of the
concepts are too simple or not constructive, like Entities/ Transformation/ Rotation/ Angle, and Audio/ Loop). Also, since the vocabulary was not focused on pervasive games, almost no special pervasive games concepts could be found in the vocabulary (did not meet Criteria 1). Other than in the MDGD area, no vocabulary was found that focused on pervasive games since pervasive games are still emerging and a commonly agreed upon definition does not exist yet [20]. In addition, some other vocabulary used for computer games was reviewed, but similar to [228], nothing met Criteria 2 since the vocabulary was usually designed for analysing game features (like [57, 155], [58], and [53]) instead of constructing game software. How the approach (both the process and the vocabulary) met these criteria is evaluated in the following two sub-sections. Since Criteria 2 was used as one of the main research decisions, the focus is on Criteria 1 and Criteria 3.

5.5.2.1 The Effectiveness as a Vocabulary

Three criteria were used when selecting such concepts to ensure that the concepts in PerGO were constructive, and with a proper level of abstraction and complexity. As a domain vocabulary, it is also very important that such concepts should be able to capture the characteristics of the target domain effectively.

The pervasive game domain is new, and in the earlier research conducted for this thesis more than 30 pervasive games were reviewed and the main characteristics were captured in one conceptual framework named TeMPS [29]. While PerGO was designed to construct a pervasive game, TeMPS was designed to understand and analyse pervasive games and therefore consists of more features, such as concepts. The effectiveness of PerGO was evaluated by examining how DSLs could use PerGO concepts to realize the TeMPS features. Among the four perspectives (Temporality, mobility, perceptibility and sociality) in TeMPS, temporality can be realized by an attribute in the Game concept in PerGO, mobility by the Map (as well as MapElement) and the MapView concepts, and perceptibility can be realized by a combination of Action concepts, Controller concepts (especially HumanPhyController and EnvironmentController) and Presenter concepts (especially PhyPre and ElePre). However, Sociality is one perspective that has not been covered well enough in the current version of PerGO, and there are plans to expand and evolve this in future work.

The six main perspectives of PerGO were proposed according to the channels where DSL concepts can be found, suggested by [17]. However, if the structure of PerGO (perspectives and the core part concepts) is in line with important frameworks and concepts in the domain of computer games, the future extension and evolution can be made easier. In [18], the authors proposed a way to automate game design by understanding it as a pipelined process: begin with an abstract game, represent it, add a thematic content “skin”, and set up control mappings. The idea of understanding game design as a pipelined process is similar to how the perspectives in PerGO are identified and organized. In PerGO, the Gameplay, AI and Virtual Game World constitute the “abstract game” since they are about the core rules of the game, and without any interactions enabled yet. While Presentation, Control, and CtrlPresentation serve to enable players to interact with games (represent and control). Narration can be thought of as the “thematic content skin” which may be extended later in PerGO. In [53], the author described orthogonal dimensions to separate the highest level game design concerns compared with those of e.g. a story or a simulation. Four dimensions
are included in this work: Simulation, Narratology, Ludology, and Gambling. PerGO supports the implementation of these dimensions. For example, simulation can be achieved by the cooperation of the virtual game world with AI as well as gameplay. Ludology is achieved by control, gameplay, and AI, and Gambling by some logic encapsulated in AI. Again, Narratology is expected to be supported in the future.

From the experience of analysing the case in section 5.4, the conclusion is drawn that the overall structure (perspectives and the core part) are clear and easy to use. Also, most of the concepts needed were already included in PerGO so that they could be used directly (38 concepts out of the overall 44 concepts).

5.5.2.2 The Efficiency as a Process

Game domain sometimes is really complex, and considerations pertaining to the various aspects of the domain weave with each other often. To keep the overall domain analysis efficient, the sequence to frame analysis should solve the underlying dependencies well. The sequence is discussed in both the procedure steps and the perspectives.

As previously introduced, in the DA process, step 1) is to go through perspectives, while step 2) is to go through concepts for common game design, and step 3) and step 4) are to go through concepts for variable game design within each perspective. This indicates two sequences - from wide to narrow (from perspectives to concepts within them), and from common to special (commonality to variability). Both are decided based on the dependency relationship.

Further, among the perspectives, an ordered sequence was suggested from Gameplay to AI, Game Element (Virtual Game World), Control and Presentation (also Ctrl-Presentation). This sequence is defined based on common game design traditions [44] [18], such as: 1) gameplay always takes the priority (whether a game can succeed depends to a large extent on the gameplay); 2) after thinking about how the player interacts with game elements (Gameplay), it is natural to think about how game elements react to the player (AI); 3) when gameplay and AI are decided, it is easier to enumerate all the world elements that could support the gameplay and AI; 4) decide how to control it and present it.

When going through concepts within a perspective, in the case where there are more than ten concepts, it is suggested to go through them according to which core concepts they are derived from. For example, Gameplay concepts should be gone through in the order of IndividualChallenge derived concepts to CollaborateChallenge derived concepts, then CompeteChallenge derived concepts and Action derived concepts. In this way, indexing and finding concepts can be more efficient.

By addressing the potential dependency among the analysis, several benefits are expected: 1) Analysers would feel natural and comfortable enough to follow the process, 2) concepts can be found in PerGO quickly, 3) it is less necessary to go back and forth to accomplish the overall analysis. As was experienced in the case study (see section 5.4), almost no situations were met where it was necessary to go back and force the domain analysis process to finish. In the overall process, the steps and perspective sequence were followed comfortably, and concepts were always found quickly.
5.5.3 The Costs and Limitations

The motivation of utilizing the proposed approach (both the DA procedure and the PerGO ontology) is to save time by reusing domain knowledge in an efficient way. This approach may not be worthy for all domains since building an ontology and evolving it also incurs costs. It was used for the pervasive game domain because it is new, and since there is no commonly agreed definition for such games. A pre-defined knowledge base helps people, especially non domain experts, to be able to design and develop such games better and more quickly. Compared to building PerGO, maintaining PerGO demanded less resources. In the earlier stages of the research, there were more times when it was felt that it was necessary to enhance the ontology structure or concepts and more resources were used to maintain PerGO. However, the ontology gradually became more mature, and by the time the case study in section 5.4 was performed, most of the maintenance work was just adding a few new concepts or changing concept names to distinguish them from each other. Such maintenance work costs very little.

However, regarding the limitations, the most prominent limitation is that PerGO is currently in the early phase of research, and all the evaluations done are based on theoretical analysis and limited experiences from internal case studies. It remains a problem as to whether or not the approach can be applied to more real world cases and others without games or MDD knowledge. Furthermore, there are some perspectives that were not included in the current version of the ontology, such as sociality and narration (as mentioned previously). Lastly, the proposed ontology and the procedure to carry out the domain analysis based on the ontology need to be put into a more complete (model driven game software development) process to evaluate its gains and costs. The plan for how to compensate for these limitations will be discussed later.

5.6 Summary

In this chapter, the Pervasive Game Ontology was proposed as the base to regulate and accelerate domain analysis for pervasive game DSL development. The usage of PerGO was demonstrated through a case study. By utilizing the proposed ontology, the domain analysis was carried out in a structured (four steps to produce outputs) and efficient way (36 out of 42 concepts in the meta-model are from the ontology in the case). The ontology based domain analysis was compared with state-of-the-art domain analysis and PerGO was evaluated theoretically. The potential user acceptance of PerGO will be evaluated in Chapter 7, together with that of GCCT.

PerGO is based on an understanding of pervasive games (through TeMPS, as illustrated in Chapter 4). PerGO aims to fill in the conceptual gap in the computer game domain (through the core part) and the pervasive game domain (through the pervasive part) (as was reviewed in Chapter 2) towards model driven game development. PerGO plays an important role in the proposed GCCT workflow (which will be illustrated in Chapter 6).
Chapter 6 Game Creation with Customized Tools (GCCT)

In this chapter, the game creation approach named GCCT is described. GCCT can be thought of as an example of applying model driven approaches in the computer game domain. An important part of GCCT is a regulated domain analysis. For pervasive games, this part can be based on PerGO, which was introduced in Chapter 5.

In this chapter, some of the technical basis is presented on which general model driven approaches can be adapted to better serve the game domain requirements. The GCCT approach is formalized and the selection of guidelines for GCCT and other game creation approaches is introduced. Case studies are performed to illustrate and evaluate the GCCT approach. Then the results are presented and the cost structure analysed. Later the benefits and limitations of GCCT are discussed from both the game development perspective and the model driven development perspective. A comparison with some related works regarding model driven game development is presented afterwards. In addition to the analytical evaluation and empirical evaluation, an evaluation was also performed with respect to potential user acceptance of GCCT, which will be introduced in Chapter 7.

6.1 Technical Basis

Nowadays, computer game development of AAA game titles is becoming more and more difficult [5]. The overall projects are becoming very big and complex, while a higher quality and shorter development cycle are expected. Instead of Game Creation from Scratch (GCS), many game developers adopt various authoring tools (e.g. [86], [77], [79]) to simplify game creation. These authoring tools provide visualized user interfaces to enable even non-programmers to specify their games, and automate the game code generation to accelerate game creation significantly. However, Game Creation with Authoring Tools (GCAT) has limitations. These tools do not address the complexity and peculiarity required by more sophisticated games or more innovative games. And they usually do not provide effective interfaces to extend/ customize features for such domains [67]. As a result, although they are popular, most users of such tools are beginners or amateurs instead of industrial professionals [67].

On the other hand, traditional game engines are more powerful and flexible. They are the state of art and mainstream tools that are used by the industry to create commercial games (especially AAA title games). Game engines can be thought of a combination of a runtime and a set of reusable modules that gather common game development foundations like entity rendering, world management and events handling. Game engines provide higher flexibility as only necessary modules are consumed to create different games. In addition, by providing in-package tools (e.g. level editors) based on the runtime and reusable modules, creating games can also be as easy as creating them in an authoring tool. However, as indicated by [67], Game Creation with
Engine and (level) Editors (GCEE) also have some drawbacks. First, game engines are usually huge, complex, and lack usability. This makes the learning curve steep and the using cost rather big, even if most of the game engine features are not used. Second, although game engines provide more freedom for customizing features, the freedom is still limited within a pre-defined game domain. Some innovative domains like pervasive games [29] and education games [242] that emerge during recent years, may not be able to benefit from existing game engines as expected.

To transcend the limitations of GCAT/ GCEE and GCS, an in-between approach named Game Creation with Customized Tools (GCCT) is proposed. With this approach, tools are created (according to actual needs), and then games are created with these tools. Model Driven (software) Development (MDD) techniques are utilized to implement this approach, as MDD claims to be able to lower the software complexity, improve the software quality, and increase the productivity of software development [9]. The practical application of MDD in many domains [243] has proven the potential and success of MDD. In the game domain, some researchers have tried to apply MDD and have achieved initial success as well ([148, 244] [15]).

MDD is claimed to bring appealing benefits, as introduced in Chapter 2. However, MDD also introduces risks. Among them, the cost issue is the most prominent one. MDD involves non-trivial costs to develop tools prior to software development. It might be hard to convince project managers to add such upfront costs to the budget, as they may not always be easy to pay off. In addition, MDD is not easy. To apply MDD, a number of practical issues must be solved to ensure the high quality of the tools and an efficient procedure, before achieving success ([117]). MDD demands a higher level of expertise from practitioners, as not everyone is used to thinking in an abstract way. Thus the degree of productivity increase is conditional. Researchers have identified some challenges to modelling languages and MDD approaches. In [108], two major challenges - abstraction and formality - were highlighted. While in [117], some other challenges like understandability and inexpensiveness were raised. These will be discussed in more detail a bit later. Before adopting MDD, some questions must be answered. For instance, under what conditions should MDD be adopted, how can the most be made of MDD in practice, and what are the main factors which impact the actual degree of productivity increase. Without satisfying answers to these questions, using MDD may be a risky choice. Whether the extra cost is worth the benefits will remain speculative until MDD is tried out.

Other researchers in Model Driven Game Development (MDGD) have not addressed these practical issues well, especially the cost issue. Most researchers have presented the evidence of productivity increase without counting the cost to develop tools, or in some cases, have not presented any real project data (this will be discussed in detail in section 6.5). This fact provided the initial motivation for the research of GCCT. To construct GCCT, general MDD approaches were enhanced by making some adaption according to game development characteristics. By doing so, better quality and efficiency can be expected.

6.1.1 A Generalized MDD Workflow

Using models to design complex software systems is not new. Models can help developers understand the problems and potential solutions through abstractions. The popu-
larity of UML [245] caused even more people to understand and accept the importance of models and design by means of a model [118]. However, due to practical reasons, when models were initially used, they often played a secondary role [117]. It was only when Model Driven Development (MDD) and Domain Specific Modelling (DSM) methods were devised to use models more advantageously that models became the primary and only artefacts 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. By providing automation, the complexity of the solution domain is hidden and full-scale code generation becomes possible. As a result of providing abstractions and code automation, more people without much domain knowledge or programming experience can write a full specification of the target system and generate the software. While the productivity is increased, the quality and maintainability of software systems are also increased [118]. MDD has been used in many domains and achieved success, according to the literature [243, 246].

The modelling language, which is usually specific to the given domain and is called the Domain Specific Language (DSL), is one of the most important artefacts that must be constructed in a MDD approach. According to [118], a DSL mainly consists of three elements: meta-model (abstract syntax and static semantics), which defines the concepts and relationship among the concepts in the language; concrete syntax, which defines what a parser for the language accepts; and the semantics, defining the meaning of the language concepts. Both the meta-model (mainly abstract syntax) and the concrete syntax need to be formally defined to further create the domain-specific editor. Further, the code generator needs to be constructed to make automation possible. Usually a code template modified from a set of working codes is used for the generator [17]. Sometimes domain specific libraries are constructed to simplify the code template programming [17]. As both languages and code generators, as well as other artefacts like the libraries, need to be constructed according to specific domain requirements, intensive domain analysis is suggested to ensure the quality of such artefacts [218]. Domain analysis usually includes commonality analysis, variability analysis, and a domain vocabulary (defining the semantics) [218]. This is also why, in many circumstances where MDD is applied, the term “Domain Specific Modelling (DSM)” is used by many researchers as well.

In addition to defining languages and other artefacts, a DSM approach should also include using such DSM artefacts to gain benefits for given domains (specifying in the editor and receiving automatically generated codes). These general DSM tasks are summarized in Table 16. Based on these tasks, researchers have defined DSM processes like [122] and [123]. Roberts and Johnson [121] described a similar recurring pattern for software development automation which can be regarded as the initial form of the DSM development process. The process consists of three phases: 1) identify a set of reusable abstractions and document a set of patterns for using them; 2) develop a runtime such as a framework to codify the abstractions and patterns; 3) define the languages and build the tools to support the runtime (such tools include editors, compilers and debuggers which automate the assembly process). Overall, such processes can be considered as a linear but iterative process which involves the tasks listed in Table 16. The GCCT approach is also based on such a general process (by making some specialization to the general tasks).
Table 16. Tasks of Domain Specific Approaches

<table>
<thead>
<tr>
<th>Model Driven Approaches</th>
<th>DSM Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Domain analysis: commonality</td>
</tr>
<tr>
<td></td>
<td>Domain analysis: variability</td>
</tr>
<tr>
<td></td>
<td>Abstract syntax definition</td>
</tr>
<tr>
<td></td>
<td>Concrete syntax definition</td>
</tr>
<tr>
<td>DSM Infrastructure Construcion</td>
<td>DSL editor construction</td>
</tr>
<tr>
<td></td>
<td>Library construction</td>
</tr>
<tr>
<td></td>
<td>Working prototype construction</td>
</tr>
<tr>
<td></td>
<td>Generator construction</td>
</tr>
<tr>
<td>DSM Use</td>
<td>Data model specification</td>
</tr>
<tr>
<td></td>
<td>(Auto) code generation (partly or completely)</td>
</tr>
</tbody>
</table>

6.1.2 Iterative Process

Due to the high domain-specific nature of MDD approaches, when requirements change, all the MDD artefacts will need to evolve as well. However, this is considered to be normal and is where the core power of MDD lies [17]. MDD approaches often start with some basic artefacts. And the number of artefacts grows increasingly with necessary iterations. When requirements change, MDD is supposed to reflect these changes in terms of artefacts/models.

Games are developed traditionally in a highly iterative way as well. Games are created for entertainment and the most important requirement of a game is to entertain players. However, it is not easy to know whether a game will entertain players or not, and make adjustments, before the game is ready to be played [34]. Thus games are often developed in an incremental way. Within each iteration of development, a small change is made and playtest is performed to consolidate the gameplay [62].

6.1.3 Combined Tasks

As pointed out by [247], models should be properly integrated in a system’s whole life cycle process to be more effective. Thus we investigate the overall game development process (instead of game software development process) to find a reasonable way to embed model driven tasks. Critical game tasks were identified which could be reused to provide inputs to the DSM definition and infrastructure building. Also, using DSM tools would facilitate game artefacts creation in return. Figure 79 presents these connection points. In the figure, orange lines indicate that traditional game tasks can be used to provide DSM inputs, while green lines indicate that using DSM provides facilitation for game tasks. These connection points served as the motivation behind formalizing a DSM process for game development, and they form the base of the GCCT approach.
Game design vs. domain analysis (Commonality): the orange line labelled “1” indicates the possible connection between traditional game design work and domain analysis work in model driven approaches. Traditional game design usually defines global settings like style, historical reference, navigation graph, classes of game objects, etc. These parts are common to all the game levels and provide primary inputs for the commonality analysis which is required by DSM approaches.

Level design envision vs. domain analysis (Variability): the orange line numbered “2” indicates that a regulated design vision above all possible levels is promising to contribute to the variability analysis. The variability analysis provides main inputs to the meta-model construction of the DSL.

Baseline prototype vs. generator template construction: When defining the template for the code generator, it is recommended to start from a full version of working codes [17]. While in the early stage of game creation, usually at least one prototype (here called the baseline prototype) is developed to verify important technical issues and consolidate the majority of the game design. The orange line with number “3” shows that this baseline prototype can be used to construct the code template.

Level design vs. data specification in editor: The line with number “4” which bridges level design and DSM is somewhat special. A traditional level design task can be thought of as specifying a game instance. With this input, DSM generates game level codes. Thus the line indicates an input from game tasks to DSM. On the other hand, level design can be specified in the game editor in a more formal and usually more user-friendly (and cheaper) way. So the line is coloured with green, indicating that DSM facilitates the game task.

Tuning prototypes vs. code generation: DSM automatically generates codes for every game (level) instance. Line “5” indicates this connection. And this is one of the main tasks where benefits are gained from DSM approaches (the other one is specifying game instances in a probably easier way).

6.2 Game Creation with Customized Tools

Model driven approaches can be adapted and applied to various domains. As an example of model driven approaches in the computer game domain, GCCT consists of a set of procedural components which connect model driven tasks and game creation
tasks. GCCT provides an extra choice for game creation in addition to GCS and other game creation approaches. In this section, the GCCT approach is formalized and guidelines provided for choosing GCCT and other approaches.

GCCT is designed primarily to create a series of games which share commonalities while differing with each other to some extent. These game series exist in different forms, however. For instance, many commercial computer games contain numerous levels. These game levels share a big part of commonality and actually constitute such a game series. In addition, games are usually developed with iterations where numerous prototypes are constructed and play-testing is performed for gameplay tuning. These prototypes can be thought of as a series of games as well. What is more, GCCT formulates not only how to develop games, but also how to develop tools. Thus we identify the following scenario where GCCT may be useful. In this chapter, our discussion is mainly based on the first scenario.

- Levels for one game
- Tuning prototypes for one game
- Game instances for a game series
- Level editors or authoring tools

### 6.2.1 GCCT Formalism

GCCT is formalized as shown in Figure 80. The overall approach indicates a linear but also highly iterative process of game development. The iterative nature comes from the requirements from both model drive development and game development, as introduced in section 6.1. In a highly iterative process, games are developed in an incremental way to enable intensive prototyping and play-testing. By doing so, the illusory gameplay can really be tried out to lower the risks of final failure. On the other hand, as tools are fully tailored to the target domain, it is quite common that they need to evolve when the target domain expands or changes. Thus an iterative and agile process is also preferred.

![Figure 80. Process View of the GCCT Approach](image)

The process consists of two linear parts: the tools creation part and the games creation part. Within them, four procedural components/tasks are involved. These are called steps. Steps 1, 2 and 3 are steps to create tools (game editor and game code generator), and step 4 is to create games with these tools. Compared with other general domain specific approaches, GCCT imposes at least two constraints: 1) a formalized
domain analysis part based on a pre-defined domain vocabulary (like an ontology); and 2) leveraging language workbench tools to automate and enhance the quality of general tasks. These tasks are explained below mainly from a game development perspective. These tasks can be thought of as the results of adapting general game development tasks to fulfil the requirements of MDD tasks as well (from the MDD perspective). The correspondence between GCCT tasks and MDD tasks is summarized in Table 17.

- **Step 1**: Customize game features that should be supported by the tools.

  Both the game editor and the code generator should be tailored according to the target game domain. Thus prior to the tool creation, the game features that should be supported need to be identified. This is formalized as a meta-model which usually is presented in a UML class diagram.

  This step corresponds to the domain analysis part and meta-model (abstract syntax) definition from the MDD perspective. Game design and levels envisioned can be adapted to contribute to this step. In GCCT, a formalized way to regulate and ease this task is proposed. In section 6.4, the domain analysis will be illustrated based on a pervasive game ontology. While in GCCT, it is only required that a domain vocabulary like an ontology should be adopted or constructed according to the actual requirements.

- **Step 2**: Customize the game editor.

  In addition to the game features, what the game editor looks like needs to be specified. Such presentation styles may be textual, tree-based, diagrammatical, graphical, or mixed. This decision can be made according to the specific needs and the technical background of the development team. Usually, the textual editor is the easiest and the most flexible one to develop. But it is less user-friendly meanwhile. In contrast, the graphical editor is able to provide the most fancy and easy-to-use interface, but also, it may be the most expensive to develop. The presentation style can be changed as the project evolves. For instance, at the beginning, textual style is used. Then part of the editor provides diagrammatical or graphical user interface as this part does not need to be changed much. Later, as the project progresses, more parts of the editor provide graphical interfaces. In addition to the style of editor, the visualized editor components need to be linked to the underlying corresponding game features.

  This step mainly corresponds to the concrete syntax definition of MDD.

- **Step 3**: Customize the code generator.

  When a game specification is received from the game editor, we need to know how to generate the game codes accordingly. This step is usually done by providing a code template. According to the code template, variable parts of the game specification can be identified, parsed, and combined with common parts coded in the template.

  This step comes from the MDD requirements. Researchers have suggested to use working code blocks to ease the definition of the generator [17]. In the research of this thesis, the proposal is to reuse the baseline prototype which usually is created during the pre-production phase of game development.
Table 17. GCCT as an Example of Model Driven Approaches in the Game Domain

<table>
<thead>
<tr>
<th>Model Driven Approaches</th>
<th>GCCT</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSM Definition</td>
<td></td>
<td>Game</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain analysis:</td>
<td></td>
<td>Feature</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commonality</td>
<td></td>
<td>Customization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain analysis:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>variability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abstract syntax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concrete syntax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSM Infrastructure</td>
<td></td>
<td>Game</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSL editor construction</td>
<td></td>
<td>Editor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library construction</td>
<td></td>
<td>Customization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Optional) library</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>construction/ adoption</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Working prototype</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>construction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generator construction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3 Code template</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>creation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4 (Auto) code generator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSM Use</td>
<td></td>
<td>Game</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data model specification</td>
<td></td>
<td>Specific</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(partly or completely)</td>
<td></td>
<td>ation and</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Auto) code generation</td>
<td></td>
<td>(Auto-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(partly or completely)</td>
<td></td>
<td>ic) Code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Step 4:** Specify game instances and generate the codes.

As game tools are expected to be currently available to use, developers should simply specify games in the editor and get the automatically generated codes from the generator.

From MDD’s perspective, this step is to specify a data model which conforms to the meta-model and generates corresponding codes. From the game perspective, traditional level design can be done in a more formalized and usually more visualized way, and codes can be generated automatically.

Table 17 shows us that many MDD tasks are performed by reusing or regulating existing game development tasks (the dark blue ones). In addition, many tasks are automated or half automated by leveraging language workbench tools (the light green ones). As a result, the extra cost incurred by involving GCCT is highly leveraged. The main mandatory tasks left are summarized as four steps and the steps are named from the game authoring perspective, as the target audience is from the game community. Steps 1, 2 and 3 are called the customization of tools instead of the creation of tools. This is because the remaining work is mainly concerned with the domain-specific customization of part of the corresponding tools. Figure 81 shows how different kinds of participants in a game development team would cooperate to develop games at a high level by utilizing GCCT.
6.2.2 Comparing GCCT and GCEE

Although the concept of MDD is relatively new, similar software reuse approaches are not new for the game domain. While MDD encapsulates common and reusable codes in domain specific libraries and generators, games use a game engine to isolate common and reusable codes (from specific codes for game objects and behaviour definition). MDD advocates providing domain specific and higher level concepts (and probably visualized interfaces) to engineers, so that they can focus more on the problem domain and specify the system solution more easily. By doing so, the overall complexity can be alleviated. Game software is complex when it comes to both the domain knowledge and the architecture. Applying MDD may help separate the two (by hiding the domain complexity in DSM artefacts).

In addition, it is quite common that the generic game engine executes specific level descriptions which are produced in a level editor. This approach of creating games using such engines and in-package level editors is known as GCEE (Game Creation with Engine and level Editor). MDD can help to achieve a similar goal: generic code patterns execute specific data (model) which is produced in a DSL editor. MDD approaches may be more structured and customized. Also, the automation degree is expected to be much higher as language workbenches ([125]) can be used to automate most of the general tasks (such as editor framework construction, linkage, compiler construction, etc.).

There are several similarities between GCCT and GCEE (Game Creation with Engine and Level Editor). GCCT can be thought of as one special way to create and use level editors. GCCT is formalized and model-based, while GCCE is more hand-
crafted. In addition, the process of GCCT may be more agile and effective. On the other hand, GCEE often provides a visualized user interface, allowing developers to specify games with high level abstractions. This set of domain-specific abstractions can be thought of as one domain specific language. When compared to creating a level editor in GCCT, GCEE modifies the existing level editor. Usually, developing a new game based on a game engine requires modifications or extensions to the level editor and probably part of the engine, according to the specific game requirements. This is because engines are usually designed and implemented for a comparatively larger game genre when using GCEE. From this point of view, the first difference between GCCT and GCEE is their different starting points. Although GCEE starts from a set of existing and working tools, this also means that the unnecessary features lead to some unnecessary learning and costs. In addition, modifying existing tools may be more expensive than creating them from scratch sometimes.

The detailed differences between GCCT and GCEE are summarized in Table 18. From Table 18, we can see that for almost all of the GCCT components we can find corresponding tasks in GCEE. The main difference is that GCCT components are more formal (as required by domain specific engineering tools), and automated (by utilizing language workbench tools).

<table>
<thead>
<tr>
<th>Table 18. Comparison between GCCT and GCEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCCT</td>
</tr>
<tr>
<td>1.1 Formalized game design</td>
</tr>
<tr>
<td>1.2 Formalized game levels envisioned</td>
</tr>
<tr>
<td>1.3 Domain model definition</td>
</tr>
<tr>
<td>2.1 Game editor UI and logic design</td>
</tr>
<tr>
<td>2.2 (Auto) game editor creation</td>
</tr>
<tr>
<td>3.1 (Optional) library construction/ adoption</td>
</tr>
<tr>
<td>3.2 Baseline prototype construction</td>
</tr>
<tr>
<td>3.3 Code template creation</td>
</tr>
<tr>
<td>3.4 (Auto) code generator creation</td>
</tr>
<tr>
<td>4.1 Level design in editor</td>
</tr>
<tr>
<td>4.2 (Auto) code generation</td>
</tr>
</tbody>
</table>

6.2.3 Selection Guidelines for Game Development Approaches

Both GCCT (Game Creation with Customized Tools) and other approaches like Game Creation from Scratch (GCS), Game Creation with Authoring Tools (GCAT) and Game Creation with game Engine and Level Editor (GCEE) provide benefits while also having limitations. GCCT is meant to provide an alternative to other approaches instead of replacing them. Selecting a proper approach for a certain project is im-
Multi-attribute utility theory (MAUT) analysis [248], where criteria are weighted according to their importance to the project, is therefore proposed for GCCT. For different domains and development teams, decision makers should evaluate and choose the most suitable approach to achieve the best success. In this section, some criteria are identified that may be useful when choosing between possible approaches.

As discussed previously, the root differences between GCS, GCAT and GCEE may be the different ways of treating (authoring) tools. While GCS does not adopt any (authoring) tools and GCAT adopts powerful but ready-made (and more fixed) tools, GCEE chooses to use more professional and highly customizable tools (level editor and engine). GCCT can be analogous to GCEE in the sense of utilizing professional and highly customizable tools. But GCCT creates these tools instead of modifying existing tools. Figure 82 presents two main dimensions where these approaches may differ from each other. In the figure, the horizontal axis indicates the customizability/maturity of the tools provided by the approaches. And the vertical axis indicates to what level of formality the tools are specified and developed. Usually, the more customizable the tools are, the less mature the tools are. GCS usually does not use general tools, and this also means it may utilize any customized tools internally. GCEE utilizes existing level editors. Quite often, some modifications are performed. As for GCAT, game developers use existing and mature tools directly without modifying the tools usually. GCCT is more flexible with respect to the customizability and maturity of the tools. Developers can decide according to the specific needs of a certain project.

On the other hand, GCCT specifies and develops tools with the highest formality. It formalizes the tools specification in the form of a meta-model, and it develops the tools following some common process supported by language workbenches (highest formality). GCEE and GCAT provide tools which usually provide some general requirements (e.g. level editor), but no explicit specification is available. The development of the tools is usually done internally and does not follow some general process (middle formality). As for GCS, it may develop tools in any way and provide interfaces that meet no external specification (lowest formality).
Table 19 identifies more differences between these approaches which can be used as criteria for decision makers when choosing among the approaches. For each criterion, a score was assigned of High (H), Middle (M), Low (L) or NA (-), indicating the performance of each approach with respect to the criterion. For instance, concerning the third criterion about dependence on 3rd tools, while GCAT and GCEE are usually highly dependent on one tool or engine, and GCCT is dependent on some general language workbench tools, GCS has the lowest dependency as it is does not depend on any (authoring) tools.

Table 19. Choosing from Game Creation Approaches: GCAT, GCEE, GCCT and GCS

<table>
<thead>
<tr>
<th>Criteria</th>
<th>GCAT</th>
<th>GCEE</th>
<th>GCCT</th>
<th>GCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using ready-made tools</td>
<td>H</td>
<td>M</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Modifying tools and use</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creating tools and use</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not using tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Small upfront efforts</td>
<td>H</td>
<td>M</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>2. Game design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Programming</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Language engineering</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>5. Quick start (Fit for small samples)</td>
<td>H</td>
<td>M</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>6. Accumulated benefits on large samples</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>7. Formality of game specification</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>8. Formality of tool specification</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>-</td>
</tr>
<tr>
<td>9. Formality of tool development process</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>-</td>
</tr>
<tr>
<td>10. Tool customizability</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>11. Small learn/use burden for unnecessary features</td>
<td>M</td>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>12. Low dependence on 3rd tools</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>H</td>
</tr>
</tbody>
</table>

6.3 Case Studies

Instead of conventional computer games, the research of this thesis focused on the pervasive game domain. Pervasive games have emerged during the last ten years. Such games involve more physical and social elements in the game, and blend the game with real life by providing the game experience all the time and everywhere. Well known pervasive games include “Mobio Threat” [100], “SupaFly” [98], “Epidemic Menace” [99], and “Capture the Flag” [249]. Although the concept of pervasive games is new, many of the features of pervasive games are applicable to a larger scope of modern computer games, including mobile games. A full-scale introduction of pervasive games can be found in [29]. Typical pervasive games are location-based, involve physical user interfaces, involve mobility, and are long lasting. In the case
studies used in this thesis, the main pervasive feature involved was location-awareness. The goals of performing the case studies included:

- To gain the practical experience of performing GCCT in general,
- To collect real project data regarding the cost (for both the tools and game software),
- To identify factors which impact the cost, and
- To gain lessons to reduce the cost.

Several rounds of preparation were performed for the case studies. During the initial phase, a few pilot projects were conducted to become familiar with model driven techniques as well as the tools and libraries that were used to implement location-awareness related features. Initial experiences yielded information as to how to perform MDD approaches in a more compact and efficient manner. As a result of these experiences, the GCCT approach was modified and consolidated. Two subsequent case studies were performed, known as RealCoins and RealPacman, to evaluate the GCCT approach. Both are location-based games. As these two case studies use similar settings, the settings of RealCoins only are presented in the following sub sections. But the cost data from both case studies will be presented and discussed in sections 6.4 and 6.5.

6.3.1 Domain Description

Below is the description of the pervasive game case RealCoins:

RealCoins 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. The mobile devices should be equipped with GPS so that the 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 the 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 the players are. In the main view of the game, a real map with information of the treasure zones, coins, and players is presented. Players are not able to see the hidden virtual coins before they or their group members enter the treasure zone where hidden coins are located. The main gameplay 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 fixed amount of coins from the player, while the resulting coins that the player can steal are 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.

RealCoins is an example of performing GCCT in location-aware computer games. The main specializations include: formalizing the game design and game levels envisioned (the domain analysis part from DSM perspective) based on a pervasive game ontology, and adopting language workbench tools in the Eclipse platform. These will be illustrated in the remaining part of this section, and Table 20 summarizes the specializations.
Table 20. RealCoins as an Instance of GCCT

<table>
<thead>
<tr>
<th>Model Driven Approaches</th>
<th>GCCT</th>
<th>RealCoins</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSM Definition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain analysis: Commonality</td>
<td>1.1 Formalized game design</td>
<td>Based on a pervasive game ontology</td>
</tr>
<tr>
<td>Domain analysis: Variability</td>
<td>1.2 Formalized game levels envision</td>
<td>Based on a pervasive game ontology</td>
</tr>
<tr>
<td>Abstract syntax definition</td>
<td>1.3 Domain model definition</td>
<td>Based on Ecore</td>
</tr>
<tr>
<td>Concrete syntax definition</td>
<td>2.1 Game editor UI and logic design</td>
<td>Based on Xtext</td>
</tr>
<tr>
<td>DSM Infrastructure Construction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSL editor construction</td>
<td>2.2 (auto) game editor creation</td>
<td>Created by Xtext project</td>
</tr>
<tr>
<td>Library construction</td>
<td>3.1 (optional) library construction/adoption</td>
<td>Adopted 3rd party libraries</td>
</tr>
<tr>
<td>Working prototype construction</td>
<td>3.2 Baseline prototype construction</td>
<td>Manually programmed</td>
</tr>
<tr>
<td>Generator construction</td>
<td>3.3 code template creation</td>
<td>Based on Xtend</td>
</tr>
<tr>
<td>3.4 (auto) code generator creation</td>
<td>4.1 level design in editor</td>
<td>Based on generated editor</td>
</tr>
<tr>
<td>DSM Use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data model specification</td>
<td>4.1 level design in editor</td>
<td>Level design in generated editor</td>
</tr>
<tr>
<td>(Automatic) code generation (partly or completely)</td>
<td>4.2 (auto) code generation</td>
<td>Automatic code generation</td>
</tr>
</tbody>
</table>

6.3.2 Domain Analysis

In this case, domain analysis was conducted based on the pre-defined ontology named Pervasive Game Ontology (PerGO) [32], which is part of this research. 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 the target DSL. All the abstractions are organized in perspectives such as ‘Challenge and Action’, ‘Virtual World Element’ and ‘Presentation’. These perspectives are helpful to frame the domain analysis as well as the concepts identification in a systematic way. Based on the perspectives and abstractions provided in PerGO, domain analysis was conducted in three steps: 1) Went through all the perspectives, identifying perspectives that related to the current domain, and recorded them; 2) Went into each perspective that had been recorded, analysed the common game design, expressed them as concepts by utilizing existing abstractions in PerGO or deriving new ones based on PerGO; 3) Went through the perspectives and concepts identified in step 2, defined and scoped the possible variability space based on the
concepts, and expressed the variability as concept attributes or relationships among
concepts. Table 21 presents part of the results of the domain analysis, as seen below.
These structured concepts and related attributes contribute to the construction of the
meta-model directly.

<table>
<thead>
<tr>
<th>PerGO Perspectives</th>
<th>Commonality (Concepts)</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Challenge and Action</strong></td>
<td>(Challenge): GetMoreCoin; (Actions): Move, GetTreasure, StealTreasure</td>
<td>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</td>
</tr>
<tr>
<td><strong>World Element</strong></td>
<td>Map, Location, Treasure, Group, Player</td>
<td>Location number, location size, location position, treasure value, treasure position, treasure number, player position</td>
</tr>
<tr>
<td><strong>Presentation</strong></td>
<td>MainView, GUI, Log-inGUI</td>
<td>Map style for client, map style for server, transparency for group members, transparency for opponents,</td>
</tr>
<tr>
<td><strong>Control</strong></td>
<td>PhysicalMove, GetTreasureByKey</td>
<td>Key for GetTreasure, key for StealCoin</td>
</tr>
</tbody>
</table>

### 6.3.3 Process and Tools

All the tools 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 have used different combinations of such projects to develop their DSM solutions for computer games [15, 158, 175]. Table 22 presents a list of tools used according to which tasks they can support in the DSM definition or DSM usage. The tasks are based on the description in [17]. The details of some of the tools will be illustrated by demonstrating their usage in the case in the remaining part of this section. To make a complete view, the domain analysis task (identifying and defining modelling concepts) is also listed in the table. Actually, it is possible to use any text editor (but better to support a table) for this task. The integration of multiple languages will not be introduced in detail since it is not used in this case. Also, maintaining the language requires cooperation among all tools instead of a specific one, and will not be illustrated in detail. The last two tasks (Domain framework construction and code automation) will also not be elaborated on because they are based on some embedded and implicit mechanisms, and developers usually do not have to manually write codes or make configurations to trigger them.

### 6.3.3.1 Meta-model Editor

As presented in Figure 83, 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 the meta-model to be drawn visually, and a tree based editor (“Sample Reflective Ecore Model Editor”, shown in the middle part) which allows the concepts
and relationships to be specified according to the aggregation tree. DSL developers can choose which editor to use according to different preferences and needs. These two editors save the meta-model separately in two files (shown in the right part of Figure 83), but changes to one of them will automatically be applied to the other when they are saved. In the case studies, the diagrammatic editor was used as it is more intuitive.

<table>
<thead>
<tr>
<th>DSM Tasks [17]</th>
<th>Eclipse Modelling Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifying and defining modelling concepts (domain analysis)</td>
<td>Text editors (can be outside of Eclipse)</td>
</tr>
<tr>
<td>Formalizing languages with meta-modelling (defining abstract syntax) by Meta-model Editor</td>
<td>“Ecore Diagram Editor” (diagrammatic), “Sample Reflective Ecore Model Editor” (tree-based) (see section 6.3.3.1)</td>
</tr>
<tr>
<td>Defining language rules by Constraints Validator</td>
<td>“Interactive OCL Console”, testing on “Dynamic Instance” (reference model/ sample model) (see section 6.3.3.2)</td>
</tr>
<tr>
<td>Integrating multiple languages</td>
<td>By importing Ecore Model in Xtext Editor (not used in this case study)</td>
</tr>
<tr>
<td>Notation for the language (defining concrete syntax) by Concrete Syntax Editor</td>
<td>Xtext [250] Editor (see section 6.3.3.3)</td>
</tr>
<tr>
<td>Testing the language by creating models in DSL Editor</td>
<td>“Sample Reflective Ecore Model Editor” (to create “Dynamic Instance” and test abstract syntax), Generated DSL Editor (to test both abstract syntax and concrete syntax) (see section 6.3.3.4)</td>
</tr>
<tr>
<td>Maintaining the language</td>
<td>Cooperation among all the tools</td>
</tr>
<tr>
<td>Generator definition by Generator Editor</td>
<td>Xtend [251] Editor (see section 6.3.3.5)</td>
</tr>
<tr>
<td>Domain framework / domain specific library construction</td>
<td>Automatically generated GenModel (generating infrastructural classes)</td>
</tr>
<tr>
<td>Code automation by generator</td>
<td>Embedded and implicit mechanism (see section 6.3.3.5)</td>
</tr>
</tbody>
</table>

Figure 83. Meta-model of Sample DSL in “Ecore Diagram Editor” and “Sample Reflective Ecore Model Editor”
6.3.3.2 Constraints Validator

Once the meta-model is decided, a model ("Dynamic instance") can be specified (prior to the concrete syntax definition) which conforms to the meta-model. Some constraints can be defined for the meta-model. These constraints are called "static semantics" by some researchers [118]. They are part of the DSL definition, but usually are specified with a different annotation than that of the meta-model. Constraints were validated which were written in OCL [168] for the dynamic instances. Figure 84 shows a typical view where OCL constraints were tested on a model: a node in the data model was selected (left 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 part of the figure).

6.3.3.3 Concrete Syntax Editor

For one domain specific language, only one meta-model/ set of abstract syntax can be defined. But it is possible to define more than one set of concrete syntax. These sets of concrete syntax conform to the same meta-model, but can support different kinds of
interface to the users. As Figure 85 shows, DSL editors based on different concrete syntax can be textual, table-based, tree-based, diagrammatical, or graphical. Each kind of concrete syntax has different strengths and limitations. For instance, textual ones may be quite flexible and cheap but less intuitive, while graphical ones are more fancy and user-friendly, but are also usually more expensive. It is also possible to change the presentation style/concrete syntax partly and gradually as the project progresses (as mentioned earlier).

In this environment, when the meta-model has been specified, it is possible to automatically generate a set of concrete syntax for the DSL. This is preferred during the early stage of a project, when the tools are more frequently (re-)constructed. Automating this step makes the overall workflow more compact and efficient. Therefore, developers can focus on the implementation of the main features. But when the tools become mature, a more user-friendly concrete syntax should be defined, according to the requirements of real target users. At that time, the generated syntax may be used as a starting point which is bug-free and working. Figure 86 shows the automatically generated concrete syntax of the DSL in this research, in the editor which was enabled by the Xtext project [250]. In this research, no other concrete syntax was defined as the case studies were only used internally. Xtext supports the definition of an abstract syntax as well, but it was not used in this case study due to the comparatively poorly visualized user interface (when compared with a diagrammatical one).

6.3.3.4 DSL Editor

After defining both the abstract and concrete syntax, the DSL editor could be generated as an Eclipse plugin and enabled in a new instance of the Eclipse IDE (also called “Eclipse”). In this new Eclipse instance, a data file written in the DSL was opened, all the keywords highlighted, and the data file then analysed and understood. This work could be done in the same Eclipse instance as well, by making some configuration to
the project. The left part of Figure 87 shows that a data file was opened in a common textual editor without any keyword highlighted. The middle part of Figure 87 shows that the same file was opened in the DSL editor with all the keywords, numbers, and strings highlighted in different colours (keywords in red, strings in blue, and numbers in grey). In the right part of the figure, we can see that when the DSL editor was ready to use, the data file can be analysed by the tree-based editor (“Sample Reflective Ecore Model Editor”).

![Sample Data in the Text Editor, DSL Editor and Sample Reflective Ecore Model Editor](image)

6.3.3.5 Generator Editor vs. Generator

As suggested in [17], a simplified process to construct the generator can utilize the reference codes available (the baseline prototype in this case) by pasting them as the entire content of generated codes, then modifying parts that contain repetition or alternatives which relate to the model data. Figure 88 shows a view of the generator editor used provided by the Xtend project [251]. Figure 89 shows that a set of files (left part) with codes was generated (right part) from the model written in the DSL (middle part).

6.4 Results

In this section, the results from the two case studies are analysed. Cost data was collected from the case studies. However, it was found that such data may float severely due to different domain complexity, developers’ expertise, and other practical issues. Thus, two cost structure models are proposed (based on the data) for more general and reasonable discussion: one from the domain perspective to help managers decide whether to adopt MDD/GCCT for a given domain, and the other from the operational perspective to help practitioners optimize important issues to actually achieve a higher
productivity increase. Some lessons learnt from the case studies are also presented, and these lessons helped lower the cost and improve the productivity.

**Figure 88. Generator Definition of the Sample DSL**

**Figure 89. Generated Codes for the Sample Data**

### 6.4.1 Result of Case Studies

Table 23 and Table 24 present the Lines of Code (LoC) and Hours (Hrs) used for the two case studies. These data were recorded as the project was carried out. For models, LoC was also recorded to quantitatively estimate the workload. The LoC of a model was got by counting the lines of code in the corresponding textual specification of the model (.ecore files in this case). This data will be used as the main evidence for the further discussion of cost and productivity increase (compared with GCS). Table 25 also presents the detailed cost for GCCT components. In Tables 23-25, the grey cells indicate the one-time costs of GCCT, while the dotted cells indicate the repetitive costs of GCCT (for each game instance). One thing to note is that, for GCCT, it was assumed that before using this approach, some (or at least one) working game prototypes should exist. Therefore, the prototypes can be reused to construct the code gen-
erator. Otherwise, if the GCCT approach is adopted from the very beginning, the cost of developing GCCT tools should be added to the cost of manually developing a game prototype.

<table>
<thead>
<tr>
<th>Table 23. Hours Used by RealCoins and RealPacman</th>
</tr>
</thead>
</table>
| ![Table 23 Hours Used by RealCoins and RealPacman](image)

<table>
<thead>
<tr>
<th>Table 24. LoC Used by RealCoins and RealPacman</th>
</tr>
</thead>
</table>
| ![Table 24 LoC Used by RealCoins and RealPacman](image)

<table>
<thead>
<tr>
<th>Table 25. Detailed Cost of RealPacman</th>
</tr>
</thead>
</table>
| ![Table 25 Detailed Cost of RealPacman](image)

LoC and Hrs are traditionally used for evaluating cost and productivity. Both LoC and Hrs have advantages and limitations. For instance, developers with weaker expertise may use many more hours than those with stronger expertise in the same domain. As a result, Hrs is often quite subjective. However, LoC also does not always objectively reflect the workload or cost. Efforts to write different kinds of codes with the same LoC may greatly differ. For example, as computer games often involve large numbers of objects, a large amount of lines of code are used to simply define these objects, and thus they are very easy to write. But the codes for artificial intelligence or path-finding algorithms are usually much more difficult to write. As a result, LoC may help us to reduce the bias brought on by different human expertise, while Hrs may help us to reduce the bias brought on by different domain complexities. For these reasons, data was recorded for both LoC and Hrs in order to try to avoid as much bias as possible.

Tables 23 and Table 24 show us that, by using GCCT, 5.4% and 6.7% of the manual development time was used for each new game instance. And the corresponding LoCs needed for GCCT were 4.67% and 25.6% of the manual ones. In other words, to develop each new game instance, using GCCT can be 4-20 times faster/cheaper than
using GCS. The cost saving was very obvious. Although there was an upfront cost for the GCCT approach, it was not very large. From the data in the tables, this extra cost to develop GCCT tools required around 60%-80% of the total time or 20%-50% of the total LoC of manually developing one game instance. That is to say, this one-time cost was paid back after two or three game instances were developed.

Despite the generally promising results, we also noticed that the data varied severely from case to case as a result of different domain complexity and practical factors. In the following sections, the cost structure is further analysed in order to try to identify such factors aiming for the best practices.

### 6.4.2 Cost Analysis from the Domain Perspective

Basically the cost consists of two parts: domain engineering costs (for the platform and including all the tools like language, editor, generator, libraries), and application engineering costs (for each specific game) [246]. The cost model used in this section is inspired by [252], but was adapted according to the scenario and practical experiences.

To aid the decision making as to whether or not MDD/GCCT should be adopted for a given domain, the total MDD cost is compared to the original cost without adopting MDD (manually writing codes) to see if it is lower. This section investigates the impacts brought on by inherent domain complexity. Thus the MDD costs are mainly structured from the domain perspective.

\[
W = C_{gcs} - C_{gcct} \\
\approx N_{ginst} * C_s - (N_{ginst} * C_m + C_{m\text{increa}} + C_{m\text{reconst}}) \\
= N_{ginst} * (C_s - C_m) - C_{m\text{increa}} - C_{m\text{reconst}} \\
= N_{ginst} * (C_s - C_m) - (C_m + I*C_p) - (C_m + R*C_p) \\
= N_{ginst} * (C_s - C_m) - (I + R)*C_p \\
\approx N_{ginst} * (C_s - C_m) - (C_v + C_e + C_r) - (I + R)*C_p
\]

- \(C_{gcs}\): Total cost of creating the game instances with GCS approach
- \(C_{gcct}\): Total cost of creating the game instances with GCCT approach (or other similar model driven approaches)
- \(N_{ginst}\): Total number of game instances that will be created
- \(C_s\): Average cost of creating one game instance with GCS
- \(C_m\): Average cost for creating one game instance with the model driven method (GCCT)
- \(C_{m\text{increa}}\): Total cost produced by (increasingly) constructing modelling tools (including routine tasks)
- \(C_{m\text{reconst}}\): Total extra cost produced by re-constructing modelling tools (including routine tasks)
- \(I\): Number of iterations that are used for tools construction
- \(R\): Number of iterations that are used for tools re-construction
- \(C_m\): Accumulated cost of constructing modelling tools
- \(C_m\): Accumulated cost of re-constructing modelling tools
- \(C_p\): Average routine cost for performing one tools (re-) construction iteration
The cost of using MDD should also include the cost of tools development. From the case studies, the coding of developing tools represents around 70% of the codes of writing one game (78.6% for RealCoins and 64.1% for RealPacman). Thus, tools development is not a trivial task. In addition, the tools development is done in an incremental way (as indicated in section 6.2). However, real experience shows that sometimes situations are met where the tools need to be somewhat re-constructed in addition to simply being extended. Assume $C_{\text{mitcrea}}$ as the total cost to increasingly construct the tools in an ideal way, and $C_{\text{mitreconst}}$ as the total cost to reconstruct the tools due to imperfect design and implementation. From this we get the formulas F1-2 and F1-3. Further, tools development often involves changes to many DSM artefacts such as the meta-model, corresponding source codes, the game editor, and generator, as well as the reorganizing of the development environment (clean, build and link). The cost of performing such a process is non-trivial even for a very small (or no) change. [117] called this cost “turnaround time” and it was thought to be even more important than the onetime cost for the full system generation. A similar experience was achieved when the case studies were performed, as such a procedure could take up to 10 minutes. This cost is something that cannot be overlooked for a small project like RealCoins or RealPacman (where modelling a game instance only takes 30-40 minutes). Given $I$ and $R$ as the number of iterations for incrementally constructing the tools and re-constructing the tools, the formulas F1-4 and F1-5 are derived. In the formula presented earlier in this section, $C_{p}$ indicates the procedural/routine cost for performing one iteration. Tasks involved in such a procedure in the environment of the case study mainly include modifying the meta-model, cleaning and generating new source codes accordingly (for the DSL concepts), setting up new xtext/xtend projects, copying and modifying the generator and data files, rebuilding and verifying.

In an ideal situation, where the development team has strong expertise and the project goes smoothly, the re-construction cost of the tools ($C_{\text{mit}}$) may be deduced to a large extent. It is assumed to be zero here to avoid distraction, but it will be illustrated more specifically in section 6.4.3. Differing from $C_{\text{mit}}$, $C_{\text{m}}$ primarily depends on the inherent scale and complexity of the target domain. $C_{\text{c}}$, $C_{\text{v}}$, and $C_{\text{r}}$ are identified as the main constituent parts of $C_{\text{m}}$. $C_{\text{c}}$ is the cost to construct tools for formalizing the commonality part of the domain (mainly in the domain library and the common part of the code template). $C_{\text{v}}$ is the cost to construct tools relating to the variability part of
the domain. This is primarily reflected in the meta-model construction. And \( C_r \) is used to indicate the cost of weaving the relationship among commonalities and variabilities within the tools (the variable part in the code template). Formula F1-6 is derived by applying these assumptions.

Examining Formula 1-6 (especially the variables highlighted) leads to some useful guidelines for deciding whether or not to adopt MDD for a certain domain/project:

- Constructing a game model must be cheaper than writing a game (\( C_m < C_s \)), because this is the only channel where benefits are gained (in the sense of cost and efficiency) by involving MDD.

- A larger number of the overall game instances within the domain is preferable. As long as \( C_m \) is smaller than \( C_s \), the benefits will accumulate for all the game instances. Thus the larger \( N_{\text{inst}} \) is, the larger the final benefits will be.

- The size and inherent complexity of the domain must be fully understood and confined to a reasonable scope. In an ideal situation, where re-constructing tools happens rarely, the one-time cost contributes to the majority of overall extra costs. There is no precise model to evaluate the domain size and complexity at present, so evaluating \( C_c, C_v, C_r \), and then summing them up may help but still relies heavily on the practitioners’ expertise and experience.

- The development team should have enough expertise to carry out the MDD iterations frequently and efficiently (with small \( C_p \)). MDD advocates an iterative process, which means the number \( I \) is designed not to be small. By doing so, the risks (of re-construction) can be lowered and the ability of adapting to emerging requirements increased. Thus, the turnaround cost of performing an iteration with small change (\( C_p \)) must be minimized to make the process lightweight and agile.

6.4.3 Cost Analysis from the Practical Perspective

Given a domain which is assumed to be suitable for adopting MDD approaches, how is it best utilized to achieve the largest productivity increase? Formulas are proposed to estimate the productivity increase by using the GCCT approach, compared with that of using GCS as F2-1 and F2-2 show. After expanding the \( C_{gcct} \) and \( C_{gcs} \) (as in the previous section), the formulas F2-3, F2-4, and F2-5 are achieved.

\[
\text{PI} = \frac{1}{C_{gcct}} - \frac{1}{C_{gcs}} \times 100\% \\
= \frac{C_{gcs}}{C_{gcct}} \times 100\% - 100\% \\
\approx \frac{(N_{\text{inst}} \times C_s) - (N_{\text{inst}} \times C_m + C_{\text{mt}} + C_t + C_{\text{reconst}}) \times 100\%}{(N_{\text{inst}} \times C_m + C_{\text{mt}} + C_{\text{reconst}}) \times 100\%} - 100\% \\
\approx \frac{C_s}{C_m + (C_{\text{mt}} + C_{\text{edi}} + C_{\text{gen}}) + (I + R) \times C_p} \times 100\% - 100\% \\
C_{\text{mt}} \approx C_{\text{mm}} + C_{\text{edi}} + C_{\text{gen}}
\]

- \( \text{PI} \): Productivity Increase
- \( C_{\text{mm}} \): Accumulated cost for re-constructing the meta model
- \( C_{\text{edi}} \): Accumulated cost for re-constructing the game editor
Further, to investigate how to achieve the best productivity increase by involving MDD, variables are investigated that may be impacted by practical issues instead of by various domains. From F2-5, we can see that, for a certain domain, $Cs$, $Cm$, $Cmt$, and $Nginst$ are mainly decided by the domain inherent characteristics. While $C'mt$, $M$, and $Cp$ (highlighted in F2-5) may depend more on how the MDD approach is performed (practical issues). And $I$ is a number that we do not intend to make small (and its size should correspond to the domain size in some manner). From experience, the cost of re-constructing the tools is assumed to mainly include three parts: the cost of reconstructing the meta-model ($C'mm$), the cost of reconstructing the game editor ($C'edi$), and the cost of reconstructing the code generator ($C'gen$). F3-1 shows the result.

F2-5 and F3-1 were investigated, and information was found which may help to improve the actual effect of applying MDD (regarding efficiency) for a given domain/project:

- The cost of re-constructing the tools ($C'mt$) must be limited to a reasonable scope. This re-construction cost is at least as important as the constructing cost ($Cmt$). $C'mt$ can be very large. When the pilot projects were performed, the tools were re-constructed frequently as features were increasingly implemented, due to poor expectations of the future requirements and casual meta-model at the beginning. As a result, the accumulated re-construction cost was several times the construction cost.

- The meta-model must be of good quality and avoid having to be re-constructed frequently during the whole project life cycle. Re-constructing the meta-model, the editor or code generator may lead to costs of different magnitudes. Both the editor and the code generator are aligned with the meta-model structure. As a result, if the meta-model is re-constructed, everything needs to be re-constructed. If this happens in the latter phase of the overall project life cycle, when most of the features have been implemented, the re-construction may resemble an extra round of construction (and increase the overall cost to a large extent). On the other hand, if the editor needs to be re-constructed to present a different user interface, then we only need to pay a higher cost for the editor (not the meta-model and generator). And the generator is in the same situation. Thus, in general, the quality of the meta-model is the key to limiting the re-construction cost.

- The number of re-constructing tools iterations ($R$) should be reduced. The combination of re-construction iterations is recommended if it does not bring other negative impacts. To lower the project risks and be more adaptable for possible changing requirements, the tools should be constructed in an incremental way. As a result, $M$ is not small (and proportional to the domain size). Different to $M$, $R$ is something we can try to minimize. When re-construction is needed, it may be preferable to put it off, and combine it with the next round of re-construction (if it is all right to continue the project with those imperfect tools).

- As an alternative to reducing $R$, reducing the turnaround cost for (re-)constructing tools with a small change ($Cp$) helps to decrease the overall cost. $Cp$ is accumulated within each iteration where tools are constructed or re-constructed. In addition, if $Cp$ is not trivial, the smooth developing experience
will be interrupted. Minimizing the procedural cost for the routine tasks helps developers to focus on the changes they have made and consequently improve the software quality. During latter phases of the case studies, such a process could be finished within 5 minutes. The iterations were performed in a more agile way.

6.4.4 Lessons learnt for improving productivity

From the case studies, practical deductions were made as to how to save cost and improve the ultimate degree of productivity increase. These lessons were supported by the cost analysis, detailed above. These are illustrated according to which variable within the cost structure (as shown in sections 6.4.2 and 6.4.3) will be decreased, in which way (with which strategy) they can be decreased (Enhance quality/ Reduce workload/ Reduce Complexity/ Automate general work), and within which part of a project these lessons should be applied (the domain analysis part, tool selection, or other practical parts). The summary of the discussion is presented in Table 11. Items A, H, and J are highlighted as they may have a global and more significant impact to the overall cost.

From Table 26, we can see that many of the lessons demonstrated the importance of the domain analysis and practical issues (including tools adoption). This is in accordance with the focus of the GCCT approach (formalizing domain analysis and regulating a compact process).

A. *Start with a stable but extensible meta-model.* This may be the most important item to make sure that the tools can be built up in an incremental way without frequent re-construction. During the pilot case studies, there was no common basic structure for the meta-models, which sometimes made it very difficult to implement new features during the incremental process. In addition, everything had to be recreated and the overall development cycle became accordingly much longer. But when PerGO was applied (in RealCoins and RealPacman) to build an extensible meta-model structure, such a situation did not frequently appear. That is why attention was paid to the theoretical base of the meta-model/ PerGO. A meta-model of good quality helps to decrease both the possibilities of tools re-construction ($R$) and the cost of re-construction ($C'mt$).

B. *Make abstractions quickly with knowledge engineering.* It is not always easy to make abstractions for a specific domain. Knowledge engineering helps developers reuse domain knowledge including domain-specific concepts and structures. Using PerGO in the case studies helped us streamline the domain analysis and define the meta-model quickly. This can help developers accelerate both the meta-model construction and the meta-model re-construction (decrease both $Cv$ and $C'mm$).

C. *Use a simple but quick editor before the requirements are finalized.* During the earlier stage of tools development, automatically generated editors were used to iterate quickly. This helps to reduce $C'edi$ and it especially helps when the editor is re-constructed due to the change from the meta-model instead of itself.

D. *Use easy-to-implement abstractions.* If it is difficult to find corresponding constructs to implement the meta-model concepts, building domain specific libraries will become much more difficult and time-consuming. In addition, specifying the relationship among them in the code template will become even more difficult. Usually, three kinds of constructs were used to implement the meta-model concepts:
data structure (e.g. for the global game configurations), procedure (e.g. for AI logic), and class (e.g. for game objects). All the abstractions in PerGO can intuitively be implemented as one of these kinds of constructs. This helped to decrease the cost of constructing ($C_r$) or re-constructing the generator ($C'_{gen}$).

E. **Reuse codes of working prototypes to construct generators.** A working starting point created the generator quicker and with less bugs. Both $C_r$ and $C'_{gen}$ can be reduced.

F. **Generate complete codes.** If some codes (even several lines of code) need to be manually written after other parts have been automatically generated, this part of the work will be repeated for all the iterations. Generating complete codes reduces the routine cost of iterations ($C_p$) and makes the overall process much more agile.

G. **Regulate the structured process within one iteration.** The process should include both game design tasks and DSM related tasks. A reasonable and regulated process helps to reuse resources in an efficient way, and shortens the routine cost for each iteration ($C_p$).

H. **Adopt highly automatic language workbench tools.** $C_p$ can be reduced as many routine tasks like generating concept classes and a default concrete syntax are automated.

I. **Adopt a highly integrated development environment.** A highly integrated development environment not only automates some parts of the tasks, but also avoids some unnecessary tasks. As a result, $C_p$ can be kept small. For instance, by copying source codes from one tool to another, or changing formats of data due to the different requirements from tools.

J. **Carry feature based/test-oriented iterations.** Within each iteration of tools (re-)construction, one or more testable features were defined. These were firstly implemented based on previous working prototypes (baseline prototype or tuning prototypes produced in previous iterations). Then each task had clear goals. And more importantly, it became possible to test and debug each feature in a timely manner. The quality of all artefacts was enhanced, and $C_{mt}$ and $C'_{mt}$ were reduced.

6.5 Discussion

In this section, the benefits and limitations of the GCCT approach are discussed, from both the game development perspective and the model driven perspective. The issues of cost and quality are addressed. Last, but not least, GCCT is compared with other model driven game development researches. And threats to the validity of the evaluation are discussed.

6.5.1 Enhanced Computer Game Development

In [87], the authors indicated that the software engineering process was not clearly understood in the computer game field, and that this hindered reliable development. By investigating factors which lead to the success or failure of computer game development, issues with transitioning from pre-production to development (called production by the author) stages can be highlighted as 1) How to transform the form of pre-production documents so that it can be used in development; 2) How to identify implied information; and 3) How to apply domain knowledge for the creative process. As a result, the authors suggested that the traditional requirements for engineering...
techniques should be improved to better support creative computer game development.

<table>
<thead>
<tr>
<th>Lessons</th>
<th>Variables</th>
<th>Strategy</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Start with solid and extensible meta-model structure</td>
<td>C’mtr</td>
<td>Enhance Quality</td>
<td>DA</td>
</tr>
<tr>
<td>B. Make abstractions quickly by knowledge engineering</td>
<td>Cv C’mm</td>
<td>Automate/ Regulate general work</td>
<td>DA</td>
</tr>
<tr>
<td>C. Use a simple but quick editor before the requirements are finalized.</td>
<td>C’edi</td>
<td>Remove unnecessary work load</td>
<td>Pra</td>
</tr>
<tr>
<td>D. Use easy-to-implement abstractions</td>
<td>Cc Cr C’gen</td>
<td>Decrease Complexity</td>
<td>DA</td>
</tr>
<tr>
<td>E. Reuse codes of working prototypes to construct generators</td>
<td>Cr C’gen</td>
<td>Enhance quality and Remove unnecessary work load</td>
<td>Pra</td>
</tr>
<tr>
<td>F. Generate complete codes (not partial)</td>
<td>Cp</td>
<td>Automate general work</td>
<td>Pra</td>
</tr>
<tr>
<td>G. Regulate a structured process within one iteration</td>
<td>Cp</td>
<td>Remove unnecessary work load</td>
<td>Pra</td>
</tr>
<tr>
<td>H. Adopt highly automatic language workbench tools</td>
<td>C’mtr Cp</td>
<td>Automate general work</td>
<td>Tool</td>
</tr>
<tr>
<td>I. Adopt a highly integrated development environment</td>
<td>Cp</td>
<td>Remove unnecessary work load, Automate general work</td>
<td>Tool</td>
</tr>
<tr>
<td>J. Carry feature based/ test oriented iterations</td>
<td>Cmt C’mtr</td>
<td>Enhance Quality</td>
<td>Pra</td>
</tr>
</tbody>
</table>

The usefulness of the GCCT approach in solving these issues is explained. Regarding the first issue, in the workflow, there is one main design document (game design in the form of commonality analysis) in pre-production and two design documents (level design in the form of variability and a set of models in DSL) in development. The commonality analysis was structuralized by PerGO perspectives (and concepts), and based on the result the variability analysis was performed (by specifying variable attributes, concepts and relationships), so the pre-production design has been used directly and efficiently in production. Both commonality and variability are in a form that is easily mapped to the meta-model elements of the DSL. For the detailed level design, since it is written in the DSL editor, the actual data must conform to the DSL syntax and semantic constraints. Therefore, the game design in pre-production is inherited naturally to the development stage. For the second issue (indicated as 2) above), it is proposed that implicit information can be identified more easily by trying to detail the domain knowledge in the same framework. The third issue is about how to promote the creative work with proper support regarding domain knowledge. In the workflow, domain knowledge has been captured and encapsulated in the DSM artefacts (DSL, libraries, and generators), so developers can focus on the creative work (instantiate the gameplay, and integrate various game objects, etc.). They use abstractions (from the problem domain) that are provided in the editor, do their creative work, and generate codes (in the solution domain).
The success of a game software depends to a large extent on whether the gameplay is really appealing. But gameplay can hardly be judged until you can really try it. This makes computer game software development tightly connected with the design (requirements), and the overall game development process is often a highly iterative process. Introducing MDD in this process helps to reuse previous design, domain knowledge, as well as implementation codes, and therefore saves a lot of human power and development time. From the perspective of MDD, researchers have also emphasized the ability for a language to evolve [17], which may rely heavily on an agile application lifecycle. In this workflow, by organically integrating DSM tasks and game development tasks, the overall process was kept iterative, in an efficient way. In a traditional game development process, the game design and prototypes are refined iteratively, while in this workflow, the game design, DSM, and prototypes are refined iteratively (DSM accelerates the prototypes, prototype and game design provides feedback to DSM).

6.5.2 Quality as a MDD Approach

In [117], the author proposed that the purpose of a model, as an essential part of engineering, is to reduce risk by better understanding a problem and its potential solutions before undertaking the expense and effort of full implementation. Such models must possess five key characteristics to be useful and effective:

- **Abstraction.** Models help us understand the essence of the subject that is modelled more easily by removing or hiding details that are irrelevant from a certain perspective;
- **Understandability.** Models present what remains (after abstracting away the details) in a form that most directly appeals to our intuition. This is a direct function of the modelling form’s expressiveness;
- **Accuracy.** Models provide a “true-to-life” representation of the system’s features of interest;
- **Predictiveness.** Models can be used to predict interesting but nonobvious properties of the system, either through experimentation or formal analysis. This is greatly dependent on the modelling form and the accuracy of the models;
- **Inexpensiveness.** Models should be significantly cheaper to construct and analyse than the system.

GCCT emphasizes these aspects of model quality as well. GCCT proposes to use a qualified and pre-made domain vocabulary to ensure the concepts included by the domain models are of a higher abstraction level (abstraction), easy to understand (understandability), and also practical to implement (accuracy). The cost structure is discussed in detail, as well as how to practically make a decision and reduce the cost (Inexpensiveness). In addition, special attention was paid to the extendability of the meta-model because the accumulated re-construction cost could otherwise be very large (Inexpensiveness). GCCT also makes it possible for interesting but nonobvious system properties to be included in various domain models (Predictiveness).

In addition, the author [118] indicated that models are of limited value if they do not fully exploit the potential for automation. Automatically generating complete programs as well as automatically verifying models were thought necessary to attain...
MDD’s full benefits, in contrast to earlier attempts that were limited to diagramming support and skeletal (and fragments) code generation. Since maintaining models and keeping models and codes consistent usually requires scarce and expensive resources, models are often abandoned once the codes have been generated. GCCT is consistent in this aspect as well. Complete code is generated and language workbench tools are utilized to enable model verification in the early stage.

Moreover, model driven development approaches are facing pragmatic challenges overall which must be overcome before they can be widely accepted by software practitioners. In particular, five issues were highlighted by the author [117]:

- **Model-level observability.** When an error is detected in the generated program, it should be easy to find the place in the model that must be fixed. As an analogy for traditional programming languages, we expect compilers at compile time or debuggers at run time to report errors;
- **Model executability.** Executable models provide us early direct experience with the modelled system;
- **Efficiency of generated code.** Efficiency (performance and memory utilization) is not an issue for the vast majority of applications. For more critical cases, the MDD system should allow for “seamless and overhead-free” embedding of the critical elements;
- **Scalability.** Tools and methods must scale up to meet large-scale industrial applications. It should be possible for hundreds of developers working on hundreds of parts of a model that are different but related. Compilation time (full-system compilation time and more important turnaround time for small incremental changes) and system size are important. But the system size does not appear to be a problem according to the experience of the largest systems to date;
- **Integration with legacy environments (including process) and systems (like tools and libraries).** This implies that not only the software, but also the development process and environment must be integrated into the legacy ones. This is to mitigate risks and also to leverage previous investments into such progress and environments.

In the GCCT approach, the usage of state-of-the-art language workbench tools not only reduces the workload to a large extent, but also ensures the quality and usability of the modelling environment (e.g. model-level observability and efficiency of generated code). In addition, GCCT imposes complete code generation. Therefore, although the models are not executable, once the models are ready, codes are generated at once. This is done to ensure that the direct experience with the system is available at once as well (model executability). Scalability is not a criteria GCCT has fully addressed. However, GCCT also emphasizes the practical issues which aim to shorten the turnaround time for small incremental changes. On the other hand, the full-system compilation time is expected to be optimized by using professional language workbench tools. As for the last of the five issues, “Integration with legacy environments (including process) and systems”, this embodies the core of GCCT. GCCT investigates the process and tasks of both DSM methods and modern computer game development traditions. Then GCCT makes connections by formalizing and reusing game tasks to provide DSM inputs, while additionally using DSM tools to ease and accelerate the construction of game artefacts.
6.5.3 Towards Minimized Risk and Cost

DSM is not cheap. DSM must pay off to be successful. Although there is no commonly agreed way to evaluate this, the best attempts were made to minimize the extra efforts incurred by involving DSM in this workflow. These efforts are discussed according to what constitutes DSM. There are three main classes of artefacts that need to be developed before DSM can be used: the DSL editor, domain specific libraries, and generators. Firstly, because the concrete syntax of DSL can be generated by some tools, the main effort that is required to develop the DSL editor is defining the metamodels. The most difficult part of defining metamodels encompasses abstractions and variability analysis. In this workflow, the existing game design task is reused, and it is structured by a pre-defined but extensible framework. From experience, this reuse does not require much extra effort in order to perform the case study, and would probably be the same for simple games. In this way, the game design can be performed without traditional descriptions in natural language and possible figures. Secondly, concerning the domain specific libraries, the efforts needed depend heavily on the abstraction and the implementation. While the abstraction efforts can be lowered by commonality analysis, the implementation efforts can be lowered by utilizing the prototype codes to a large extent as well. Lastly, the generator reuses the prototype codes directly. This especially applies the first time the prototype is constructed; then the language developer is recommended to paste workable code snippets into the generator templates, and then replace the variable parts and further refine them [17].

![Cost Model for a Mixed Approach](image)

However, some may argue that, for some projects, the risks are more critical than the costs. Actually this is true for most commercial game projects. And that is why game projects often utilize intensive iterations where numerous prototypes are developed for play-testing. Model driven approaches share similar concerns. Researchers have advocated performing MDD approaches in familiar domains [17]. And it is not easy to get familiar with a domain before having actually programmed several products. As an alternative, a mixed and evolving process may be more practical for many projects. Figure 90 presents how such a process may look like. In the figure, the horizontal axis indicates time or game instances that have been developed, while the vertical axis indicates the LoCs or Hrs used. The red curve and the blue curve indicate how the cost for each game instance changes when more game instances are developed for GCCT and GCS separately. In the left part of the figure, we can see that although GCCT requires more upfront costs at the beginning, the cost per game instance may drop more quickly than that of GCS as more game instances are developed. The cost per game instance can be much less than that of GCS, since the tools are mature at that time. The specific positions of break-even points (where the two curves intersect, indicating
that the upfront cost of GCCT has been paid off) may differ for different domains and projects. In a mixed approach, managers may prefer not to pay for the upfront cost at the beginning. However, as the domain is further understood and a balance between approaches is achieved over time, managers would likely benefit from model driven approaches (as the middle part of the figure shows). As a result, the upfront cost is postponed in the middle of the project rather than the beginning (the right part of the figure). For a new game domain, it may be preferable for game developers to write several game instances first, familiarize themselves with the domains, and then evaluate whether MDD is still a proper option, before really beginning to use MDD. By doing so, the cost of involving MDD may not be minimized, but it may still be a good option as long as it is cost efficient for the rest of the project.

### Table 27. Comparing GCCT with Other MDGD Research

<table>
<thead>
<tr>
<th>Domain Analysis Approach</th>
<th>Adapted Process</th>
<th>Real Cost Data</th>
<th>Cost Structure Analysis</th>
<th>Usage Scenario (When)</th>
<th>Efficiency Tips (How)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[228, 244] ([67, 240])</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[148] ([163], [164])</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[147, 159]</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[15]</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[126]</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[11]</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[13]</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[151]</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[175]</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GCCT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

### 6.5.4 Comparison with Other Model Driven Game Development Research

Some researchers have presented the trials faced when applying model driven techniques in the game development domain. However, most of these techniques were presented primarily as MDGD samples for specific domains without much customization and rigorous analyses of modelling experiences. In most cases, neither the DSLs nor the approaches were general enough to be used elsewhere. In addition, when considering the productivity increase, only a few researchers presented data for LoCs and Hrs without counting the cost of tools development, and even fewer analysed the cost structure in depth. No papers were found that discussed further issues regarding when to adopt the MDD approach for game development, or how to use the MDD approach to achieve the best productivity increase. No intensive research was found addressing the approach of detailed domain analysis within the model driven game development domain. Table 27 summarizes the differences found between GCCT and other MDGD research.
6.5.5 Threats to Validity

Some of the threats to the validity of the case studies and research of this thesis are discussed. The concepts of internal validity, external validity and construct validity, defined in [253], are used as a basis for the discussion.

Construct validity is about the relationship between theory and observations. LoC and working hours were recorded as the main evidence for the cost analysis and efficiency discussion. However, both LoC and Hrs may cause bias. The two case studies were performed to test the applicability of GCCT. The case studies and corresponding discussions mainly focused on the efficiency of GCCT. However, there may be other methods, like user acceptance testing, which should have been used. Another round of evaluation has been conducted, and the results should be reported in the near future.

Internal validity is concerned with the treatment and the outcome. GCCT requires comprehensive knowledge and techniques. The case studies were performed internally, and therefore there was no way to judge how different the result would have been if others without a similar technical background had performed the same case studies. In addition, in the case studies, due to limited resources and time, few iterations were performed for tools reconstruction. However, to reduce bias, the data from the case studies (Tables 23 and 24) was not directly used to conduct any quantitative analysis. Instead, the cost structure was modelled based on the analysis of the data from the case studies. Analysing the cost structure appeared to be a more objective approach, and this supported the practical experiences in reducing cost.

External validity is about to what extent the study result can be generalized to other situations and other people. To validate the applicability of the GCCT approach, it was applied in the two case studies: RealCoins and RealPacman. These applications can be thought of as instances of the general GCCT approach (as shown in Table 20). The main specialization of this approach is that it uses a pervasive game ontology for the general domain vocabulary, and uses the Eclipse platform for the general language workbench tools. The results met the expectation. But for other game domains (like non-pervasive game domains), and other people (especially those without a model driven background), an open question is raised as to whether or not GCCT would work well in the same way. An evaluation is being planned to include a larger scope of users with different technical backgrounds, to take this into consideration.

6.6 Summary

In this chapter, the enhanced model driven approach GCCT was introduced to improve computer game development. In GCCT, a set of procedural components were proposed based on using existing game tasks and language workbenches. Case studies were performed to investigate the performance of GCCT. The result shows that GCCT greatly accelerated the game development (4-20 times), and the upfront cost could be paid off after developing several game instances (2-3). Despite the promising result, GCCT should not be regarded as a silver bullet for every project and development group, due to the high technical threshold and upfront costs. Guidelines were identified to ease the selection between GCCT and other game development approaches. The cost structure was also analysed to support further decision making in order to achieve the best practical results. In addition to increased productivity, proper
use of the GCCT approach can also provide smoother communication, improved quality, and better balance between risks and costs for computer game development. A comparison with other model driven game development approaches indicates that GCCT addressed more critical issues regarding applying model driven approaches in the game domain (e.g. domain analysis and overall process), and provided a fullerscale discussion with respect to the usage scenario and practical guidance.

An important part of GCCT consists of a regulated and efficient domain analysis. For pervasive games, this part can be based on PerGO (as introduced in Chapter 5). In addition to the internal evaluations performed on PerGO and GCCT separately, a combined survey was performed to evaluate potential user acceptance of GCCT and PerGO. The survey and its results are presented in Chapter 7.
Chapter 7 User Acceptance of PerGO and GCCT

This chapter introduces the empirical research regarding how PerGO and GCCT might be accepted by potential users. This research was performed by one combined survey because PerGO and GCCT are tightly connected to each other. Introducing and evaluating them together is more reasonable and practical than doing it separately. The main contents of PerGO and GCCT (as well as the individual evaluations) have been introduced in Chapter 5 and Chapter 6.

This chapter firstly introduces the hypothesis and method of this empirical research. Then the settings are described for the survey and the background information of the respondents. Later the initial results are presented, and the reliability and validity of the data tested. The test data is analysed with respect to the defined hypothesis. Lastly, the test results are discussed and the chapter summarized.

7.1 Approach

The evaluation method is based on the Goal, Question Metrics (GQM) approach [254]. According to the GQM paradigm, a research goal is first defined at the conceptual level, then a set of research questions is defined at the operational level, and finally a set of metrics is listed which are used to answer the research questions. The research goal of this research was defined as follows:

*The purpose of this study was to evaluate the acceptance degree and to explore potential impacts to the acceptance of PerGO and GCCT from the point of view of potential game developers.*

Two research questions were defined by deconstructing the research goal.

- RQ1: Will PerGO and GCCT be accepted by potential users?
- RQ2: Will game and MDD background hinder the user acceptance of PerGO and GCCT?

For RQ1, constructs will be investigated as defined by the Technology Acceptance Model (TAM) [26] which is a widely used research model to test the attitude and intention of users to adopt new technologies. TAM has defined three main constructs: Perceived Usefulness (PU) as the extent to which potential users believe that using the technology will improve the job performance; Perceived Ease of Use (PEOU) as the extent to which potential users believe using the technology will be free of effort; and Intention to Use (IU) as the strongest predictor of usage behaviour. TAM claims that IU can be explained by PU and PEOU. In addition, PU is determined by PEOU. Figure 91 presents the overview of TAM variables and expected relationships.
Based on the TAM model, the following hypotheses were derived for RQ1:

- **H1-1** For PerGO, perceived usefulness will have a significant influence on IU.
- **H2-1** For PerGO, perceived ease of use will have a significant influence on IU.
- **H3-1** For PerGO, perceived ease of use will have a significant influence on perceived usefulness.
- **H1-2** For GCCT, perceived usefulness will have a significant influence on IU.
- **H2-2** For GCCT, perceived ease of use will have a significant influence on IU.
- **H3-2** For GCCT, perceived ease of use will have a significant influence on perceived usefulness.

Further, regarding the potential impact on user acceptance from the technical background of the potential users, three aspects will be investigated: pervasive game concepts, the game development experience, and MDD expertise. As PerGO is designed for the pervasive game genre, people with some basic knowledge regarding pervasive game concepts may more easily understand the ontology (H4). On the other hand, it is expected that these people would find that the extensive and full-scale vocabulary could help them to define pervasive game features as they know more about this emerging and complex domain (H5). In addition to pervasive game concepts, it is of interest to find out whether the general computer game development experience would influence how a person accepts PerGO and GCCT. Will game developers find PerGO and GCCT more useful than amateurs or non-developers (H6, H7)? What is more, as the overall approach is adapted from the general MDD approach, it is of interest to find out if a lack of MDD expertise would hinder the understanding and perceived ease of use of PerGO and GCCT (H8, H9).

- **H4** Those familiar with pervasive game concepts interpret PerGO to be more useful.
- **H5** Those familiar with pervasive game concepts interpret PerGO to be easier to use.
- **H6** Those familiar with computer game development interpret PerGO to be more useful.
- **H7** Those familiar with computer game development interpret GCCT to be more useful.
- **H8** Those familiar with MDD interpret PerGO to be easier to use.
- **H9** Those familiar with MDD interpret GCCT to be easier to use.

Surveys are used as the instrument for TAM. TAM came with a set of pre-defined questions for the three constructs. These questions/measures (six questions for PU and six questions for PEOU) were refined and validated in [26]. The two six-item scales exhibited high convergent, discriminant and factorial validity. Based on these items,
the majority of the questions in this survey were developed. The survey was complemented by other questions regarding technical background etc., and the survey was distributed to approximately 50 persons in Norway. IBM SPSS Statistics 21 [255] and SmartPLS 3 [256] were used as the statistics tools for analysing the data. Cronbach’s alpha coefficient was tested for each construct with regards to the internal consistency reliability. Correlation and regression tests were used to validate the TAM models (H1-3 for both PerGO and GCCT). Correlation tests were used for H4-H8.

7.2 Settings for the Survey
The selection criteria used to find proper test subjects are important. As PerGO and GCCT are not designed for amateurs, understanding and using them may be difficult for persons without the necessary expertise. The intention was to recruit students with at least some basic programming language and modelling language knowledge, preferably those with either MDD or game development experience. As most of the recruitment happened within NTNU, the education level of the participants was comparatively high (most were at least bachelor students). As a result, 47 persons were recruited to take the survey, and 46 of them answered all of the questions (one person left answers to some questions blank).

As PerGO and GCCT are tightly connected to each other, one survey was conducted for both of them. In addition, since PerGO and GCCT are not ready-to-use tools, two videos were developed to introduce PerGO and GCCT to the participants so that they could understand the approaches before answering questions, instead of allowing participants to try the approaches out. Both videos are Prezi [257] presentations which introduced and illustrated the basic ideas of GCCT (video1) and PerGO (video2) based on the RealPacman games sample (as mentioned in Chapter 6). The videos were 15 minutes long in total. MDD terminologies were removed (like domain analysis, domain specific language, etc.), as well as abstruse game terminologies (like game engine, level editor, etc.) in order to ease the understanding of participants. As a result, some internal testing showed that participants with a preliminary knowledge of programming and modelling would easily understand the essence of GCCT and PerGO by watching the two videos.

As mentioned previously, for the questions used in the survey the majority were developed based on pre-defined questions which came with the TAM model (as these have already been validated in terms of statistical performance). By adapting the usage scenario and subjects, 14 questions were created (6 for PU, 6 for PEOU, and 2 for IU) for PerGO and GCCT separately. These were complemented with some questions regarding general personal information as well as the participant’s technical background in games and models. The complete survey is shown in Appendix A.

The actual person recruitment and data collection procedure consisted of three parts. Firstly, the videos and questions were tested among several colleagues with different technical backgrounds (games, MDD, or only preliminary programming and modelling knowledge). The feedback was collected and used to revise the videos for smoother illustration and easier understanding. The testers did not formally answer the questions during this stage. Secondly, a small scale survey was performed in a model driven course at NTNU. In the course, related model driven techniques were taught and the Eclipse platform was used for exercises. Thus, the assumption was
made that students in the course would have a comparatively better knowledge of MDD. The two videos were played on the spot. 14 volunteers (13 with complete answers to all questions) participated and filled out the paper-based survey (Appendix A). Finally, more participants were recruited from various channels (such as master students who were enrolled in game development courses, and Ph.D. students with a computer science, information system, or similar background). Both the videos and questions (same to those presented in Appendix A, but in a form which is friendlier to web users) were provided online so that participants were able to watch the videos and answer the questions by themselves when they had time. As a result, 32 out of a total of approximately 60 persons answered the survey. Thus, in total, 46 responses were included in the data analysis.

### 7.3 Demographics of the Respondents

The respondents were recruited mainly from universities, and mostly from those with an information technology background. Figure 92 shows the demographics of the respondents regarding age, sex, education and whether they plan to develop games in the near future. From the figure, we can see that the majority of respondents were in the age groups 20-29 (43%) and 30-39 years old (46%). This corresponds to the overall high degree of education degree of the respondents. More than half of them were Ph.D. students or with higher education. The gender distribution was male dominant (61%). And more than half (65%) of all respondents indicated that they plan or probably plan to develop games.

![Figure 92. Demographics of the Respondents](image)

In order to explore whether specific domain knowledge would influence understanding and further acceptance, survey participants were also asked to provide basic information about their knowledge regarding games and models. Figure 93 presents the information collected from the respondents. From the figure, we can see that most of the participants have some knowledge of computer game development. Among the total of 46 respondents, 35 indicated that they knew something about how to develop computer games, and another 2 were very familiar with this topic. Regarding pervasive games, around half of the respondents said they knew something (19) or were very familiar (3) with the topic. All respondents knew UML and most of them (34) were very familiar with it. The distribution of MDD knowledge is comparatively even. 12 respondents were very familiar with MDD, 19 had some knowledge, while another 12 knew nothing about MDD.
7.4 Descriptive Result

The mean values for the TAM constructs are summarized in Table 28. From the table, we can see that, in general, both PerGO and GCCT are acceptable, as the mean values of all the key constructs (PU, PEOU, and IU) are above 3.0. Overall, respondents seemed to prefer PerGO a little more than GCCT (no one chose the minimum value for both PU and IU). However, the mean values of IU are almost the same (3.75).

For both PerGO and GCCT, the mean values of PU are very close to those of IU, especially for GCCT (3.79 for PU and 3.75 for IU). But the mean values of PEOU are lower than both PU and IU. More detailed results regarding all items can be found in Appendix B.

Table 28. Mean Values of TAM Constructs

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std.Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PerGO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PU1</td>
<td>46</td>
<td>2.0</td>
<td>5.0</td>
<td>3.9</td>
<td>.75879</td>
</tr>
<tr>
<td>PEOU1</td>
<td>46</td>
<td>1.0</td>
<td>5.0</td>
<td>3.37</td>
<td>.81222</td>
</tr>
<tr>
<td>IU1</td>
<td>46</td>
<td>2.0</td>
<td>5.0</td>
<td>3.75</td>
<td>.87235</td>
</tr>
<tr>
<td>GCCT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PU2</td>
<td>46</td>
<td>1.0</td>
<td>5.0</td>
<td>3.79</td>
<td>.65869</td>
</tr>
<tr>
<td>PEOU2</td>
<td>46</td>
<td>1.0</td>
<td>5.0</td>
<td>3.45</td>
<td>.68301</td>
</tr>
<tr>
<td>IU2</td>
<td>46</td>
<td>1.0</td>
<td>5.0</td>
<td>3.75</td>
<td>.90523</td>
</tr>
</tbody>
</table>

7.5 Test of Measures

Before using correlations and regression analysis to validate the TAM models for PerGO and GCCT, the reliability of the scales was first verified. The inter-item reliability was computed for each construct of PerGO and GCCT. This was measured by
Cronbach’s alpha, which is a reliability coefficient. Cronbach’s alpha is based on the average covariance among scale items. Cronbach’s alpha ranges in value from 0 to 1.

The higher the Cronbach’s alpha is, the higher the reliability of the scale is. Negative alpha can also occur, which means that the items in question are not positively correlated, and the reliability model is violated. From Table 29, we can see that all the Cronbach’s alpha values for our scales are higher than 0.8, indicating a good reliability of the scales.

<table>
<thead>
<tr>
<th>Table 29. Internal Consistency of TAM Constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>PerGO</td>
</tr>
<tr>
<td>PU1</td>
</tr>
<tr>
<td>PEOU1</td>
</tr>
<tr>
<td>IU1</td>
</tr>
<tr>
<td>GCCT</td>
</tr>
<tr>
<td>PU2</td>
</tr>
<tr>
<td>PEOU2</td>
</tr>
<tr>
<td>IU2</td>
</tr>
</tbody>
</table>

7.6 Test of TAM Related Hypotheses (H1- H3)

In the TAM model, it was indicated that PU and PEOU might be the fundamental determinants of user acceptance. In addition, PEOU might be a causal antecedent to PU. To validate the TAM related hypotheses defined previously, for both PerGO and GCCT, correlation was first used to measure the linear relationship between their corresponding variables. Regression analysis was then used to further validate their relationship.

7.6.1 PerGO

The correlation between PerGO variables was calculated, and the results are presented in Table 30. For PerGO, we can see a high correlation between each pair of TAM constructs.

<table>
<thead>
<tr>
<th>Table 30. Correlation of TAM Constructs for PerGO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>PU1</td>
</tr>
<tr>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>PEOU1</td>
</tr>
<tr>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>IU1</td>
</tr>
<tr>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

*. Correlation is significant at the 0.05 level (1-tailed).

**. Correlation is significant at the 0.01 level (1-tailed).
Further, regression was used to investigate the relationship between PU, PEOU, and IU. Linear regression can estimate coefficients of a linear equation, involving one or more independent variables which best predict the value of the dependent variable. It was of interest to determine if PU and PEOU could predict IU. When PU and PEOU were set as independent variables, and IU as the dependent variable, the regression tests showed that only PU is a predictor. The relationship between PEOU and IU is not statistically significant. It was also of interest to determine if PEOU could predict PU, and this test was carried out by setting PEOU as the independent variable and PU as the dependent variable. The results showed that PEOU is a predictor of PU. The detailed regression results for PerGO can be seen in Appendix C. Based on the regression analyses, the TAM model for PerGO was drawn (see Figure 94). The figure summarizes the coefficients (labelled on the lines), t-significance (as stars), and R² values (in the boxes of IU and PU). The model explained more than 60% of the variance in IU and more than 70% of the variance in PU.

![Figure 94. Structured TAM Model for PerGO](image)

According to the regression data, hypotheses H1-1 and H3-1 are supported for PerGO, while H2-1 is not supported (see Table 31).

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>T value</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported</td>
<td>H1-1 PU1→ IU1</td>
<td>3.181</td>
</tr>
<tr>
<td>Not Supported</td>
<td>H2-1 PEOU1→ IU1</td>
<td>1.453</td>
</tr>
<tr>
<td>Supported</td>
<td>H3-1 PEOU1→ PU1</td>
<td>10.669</td>
</tr>
</tbody>
</table>

### 7.6.2 GCCT

Similar to PerGO, the correlation was calculated between each two constructs for GCCT. The results in Table 32 show high correlations between them, especially PU and PEOU.

Regression analysis was then used to investigate the relationship between PU, PEOU and IU for GCCT. Appendix D presents the detailed results, and Figure 95 summarizes coefficients (labelled on the lines), t-statistics (in parentheses), and R² values (in the boxes of IU and PU). The data shows that for GCCT, PU is a predictor of IU (as expected), but PEOU is not a predictor of IU. In addition, PEOU was found to be a predictor of PU, as expected. The model in Figure 95 explained around 70% of the variance for both IU and PU in GCCT.
The results show that hypotheses H1-2 and H3-2 are supported. But H2-2 was not supported with statistical significance. Table 33 summarizes the validation results of the hypotheses.

Table 32. Correlation of TAM Constructs for GCCT

<table>
<thead>
<tr>
<th></th>
<th>PU2</th>
<th>PEOU2</th>
<th>IU2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU2</td>
<td>.857**</td>
<td>.820**</td>
<td></td>
</tr>
<tr>
<td>Sig. (1-tailed)</td>
<td>.000</td>
<td>.000</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>46</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>PEOU2</td>
<td>.857**</td>
<td>1</td>
<td>.782**</td>
</tr>
<tr>
<td>Sig. (1-tailed)</td>
<td>.000</td>
<td>.000</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>46</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>IU2</td>
<td>.820**</td>
<td>.782**</td>
<td>1</td>
</tr>
<tr>
<td>Sig. (1-tailed)</td>
<td>.000</td>
<td>.000</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>46</td>
<td>46</td>
<td>46</td>
</tr>
</tbody>
</table>

*. Correlation is significant at the 0.05 level (1-tailed).
**. Correlation is significant at the 0.01 level (1-tailed).

Table 33. Summary of TAM Related Hypotheses for GCCT

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>T value</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported</td>
<td>H1_2 PU2-&gt; IU2</td>
<td>3.463</td>
</tr>
<tr>
<td>Not Supported</td>
<td>H2_2 PEOU2-&gt;IU2</td>
<td>1.829</td>
</tr>
<tr>
<td>Supported</td>
<td>H3_2 PEOU2-&gt;PU2</td>
<td>11.015</td>
</tr>
</tbody>
</table>

7.7 Test of Additional Hypotheses (H4- H9)

Spearman correlations were used to look at whether domain knowledge influences user acceptance for both PerGO and GCCT. The domain knowledge investigated included three parts: pervasive game concepts, computer game development experience, and MDD expertise.

7.7.1 Game Domain Knowledge

Pervasive game concepts and general computer game development knowledge are potential factors which may influence the acceptance of both PerGO and GCCT. PerGO is especially designed for the pervasive game genre, which is emerging and relatively unknown. It has been hypothesized that people who have some knowledge of PerGO related concepts may find it easier to understand/ use (H5). In addition, they
may find PerGO more valuable as they know the complexity of this domain (H4). Spearman correlations were used to test H4 and H5. The results are shown in Tables 34 and 35.

- **H4** Those familiar with pervasive game concepts interpret PerGO to be more useful.
- **H5** Those familiar with pervasive game concepts interpret PerGO to be easier to use.

### Table 34. Correlations between Pervasive Game Concepts and PU1

<table>
<thead>
<tr>
<th><strong>H4</strong> (Supported weakly)</th>
<th>PU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pervasive Game Concepts</td>
<td>Pearson Correlation</td>
</tr>
<tr>
<td></td>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td><strong>(Supported weakly)</strong></td>
<td></td>
</tr>
<tr>
<td>Pervasive Game Concepts</td>
<td>Pearson Correlation</td>
</tr>
<tr>
<td></td>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

* Correlation is significant at the 0.05 level (1-tailed).
** Correlation is significant at the 0.01 level (1-tailed).

### Table 35. Correlations between Pervasive Game Concepts and PEOU1

<table>
<thead>
<tr>
<th><strong>H5</strong> (Supported weakly)</th>
<th>PEOU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pervasive Game Concepts</td>
<td>Pearson Correlation</td>
</tr>
<tr>
<td></td>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td><strong>(Supported weakly)</strong></td>
<td></td>
</tr>
<tr>
<td>Pervasive Game Concepts</td>
<td>Pearson Correlation</td>
</tr>
<tr>
<td></td>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

* Correlation is significant at the 0.05 level (1-tailed).
** Correlation is significant at the 0.01 level (1-tailed).

From Tables 34 and 35, we can see that both H4 and H5 are supported, with weak statistical significance. This was as expected, and the results are further supported by the different mean values of PU1 and PEOU1 which correspond to persons with different pervasive game knowledge. From the mean values shown in Table 36, we can see that persons with some (or more) knowledge about pervasive games provided in average 0.35 and 0.32 more scores than those who know nothing about pervasive game.

### Table 36. Mean of PU1 and PEOU1 Corresponding to Different Pervasive Game Knowledge

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Pervasive Game Knowledge</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU1</td>
<td>24</td>
<td>Nothing</td>
<td>3.7217</td>
<td>.61522</td>
<td>.12558</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Something+</td>
<td>4.0750</td>
<td>.66590</td>
<td>.14197</td>
</tr>
<tr>
<td>PEOU1</td>
<td>24</td>
<td>Nothing</td>
<td>3.2921</td>
<td>.53907</td>
<td>.11004</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Something+</td>
<td>3.6132</td>
<td>.78963</td>
<td>.16835</td>
</tr>
</tbody>
</table>

In addition to pervasive games background, general computer game development experience may bring impacts to the user acceptance as well. The assumption was made that the level of computer game development experience would probably lead to different understandings of the usefulness of both PerGO and GCCT. People who know how to develop computer games might compare the approaches they have tried before with this approach. By doing so, they could achieve a considerable and more rational understanding of this approach, and decide whether it is really useful. On the other hand, people without prior computer game development experience might try to com-
pare this approach with some general application development IDE, or intuitively provide a more emotional score. The correlation between game development experience and PU1, as well as PU2, was calculated. However, as Tables 37 and Table 38 show, the correlations were not significant. This indicates that the computer game development experience did not obviously influence how participants perceived the usefulness of both PerGO and GCCT.

- H6 Those familiar with computer game development interpret PerGO to be more useful.
- H7 Those familiar with computer game development interpret GCCT to be more useful.

### Table 37. Correlations between Game Development Experience and PU1

<table>
<thead>
<tr>
<th>H6 (Not supported)</th>
<th>PU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game Development Experience</td>
<td>Pearson Correlation</td>
</tr>
<tr>
<td></td>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

* Correlation is significant at the 0.05 level (1-tailed).
** Correlation is significant at the 0.01 level (1-tailed).

### Table 38. Correlations between Game Development Experience and PU2

<table>
<thead>
<tr>
<th>H7 (Not supported)</th>
<th>PU2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game Development Experience</td>
<td>Pearson Correlation</td>
</tr>
<tr>
<td></td>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

* Correlation is significant at the 0.05 level (1-tailed).
** Correlation is significant at the 0.01 level (1-tailed).

#### 7.7.2 Model Domain Knowledge

As GCCT is derived from general MDD approaches, and PerGO is used for important domain analysis tasks which are required by MDD approaches, there is concern that people with proper MDD knowledge will not accept PerGO and GCCT as these do not appear to be easy to use. Complex terminologies were removed in the introduction video, and the approach was explained from a purely game development perspective. Further, the correlation between MDD knowledge and PEOU was tested for both PerGO and GCCT. As Tables 39 and Table 40 show, the correlations are not obvious indicating that the MDD knowledge did not bring significant impact to the user attitudes.

- H8 Those familiar with MDD interpret PerGO to be easier to use.
- H9 Those familiar with MDD interpret GCCT to be easier to use.

### Table 39. Correlations between MDD Knowledge and PEOU1

<table>
<thead>
<tr>
<th>H8 (Not supported)</th>
<th>PEOU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Driven Knowledge</td>
<td>Pearson Correlation</td>
</tr>
<tr>
<td></td>
<td>Sig. (1-tailed)</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
</tbody>
</table>

* Correlation is significant at the 0.05 level (1-tailed).
** Correlation is significant at the 0.01 level (1-tailed).
### 7.8 Discussion

In this section, the results presented in the previous sections are discussed as well as some threats to validity.

#### 7.8.1 Discussion of Test Results

The results presented in section 7.4 indicate that overall, both PerGO and GCCT were perceived as useful to the participants. On average, a score of 3.9 was given to PU for PerGO and 3.79 was given to PU for GCCT. Although the average score for ease of use was perceived to be slightly lower (3.37 for PEOU1 and 3.45 for PEOU2), the average score for intention to use was higher than the middle score (3.75 for both). In the survey, there was also an open question for general comments. Thirteen participants left comments which were of use to help explain the scores. Among these 13 participants, 6 of them thought of the approach as “interesting”, “cool”, “good”, or they “like” it. Especially, GCCT was thought to be a choice for “efficiently creating lightweight casual games” (compared with other heavy tools that are available at present). Also, PerGO might help game developers to “concentrate on key values of the game” (by providing pre-defined high level abstractions). In addition, PerGO might give “a comprehensive understanding of domain knowledge for beginners”, thus becoming useful for “teaching potential talented game developers”. As the TAM models in section 7.6 show, the perceived usefulness supported the final intention of use for both PerGO and GCCT. This indicates that respondents might like to use PerGO and GCCT as they are useful for their work.

On the other hand, the average scores of PEOU were not as high as those of PU for both PerGO and GCCT. In addition, PEOU did not predict IU as we can see from the TAM models in section 7.6. Three potential reasons are considered with respect to why PEOU did not predict IU. Firstly, both PerGO and GCCT are not easy to use by themselves. PerGO involves more than one hundred terminologies related to computer games or pervasive games, and it is not possible to grasp and think of them easily just by watching 20-minute videos. GCCT stems from technology intensive MDD approaches, and it is not some ready-made tool providing easy but simple features. Actually, the intention never was to make PerGO or GCCT to be just “easy” to use. The intention was to make PerGO and GCCT easy in comparison to general MDD approaches. However, this was not reflected in the video (as an attempt was made to avoid the scenario where people without a background in MDD would have difficulties understanding the content). One participant also commented that there should be something to “compare to” when talking about ease of use. Secondly, due to limited time and resources, participants were not able to really try out GCCT and PerGO. And to ensure that all participants perceived the subjects with the same information, ready-made videos were chosen for both on-site participants in the model course and all oth-
er online participants. No live demo was provided which could have otherwise illustrated the approach more directly. This may bring bigger impact to PEOU than PU. As a result, participants could not estimate how easy the approach is by participating in real operations. Instead, they assumed how easy the approach is by watching the demonstration. This issue was reflected by comments as well. Several participants commented that they were not “certain”, it is hard to give “decisive answers”, and “some of the questions cannot be answered objectively” without “trying the tools”. 

Thirdly, as understanding PerGO and GCCT requires some preliminary knowledge, like programming, and filling out the survey required at least twenty minutes, it was not easy to perform this survey on a large scale with many people. An attempt was made to recruit in total 70/80 participants, but only 46 valid respondents were found. The small sample size may also bring uncertainty to the results. 

From the results shown in section 7.7.1, a weak correlation was found between pervasive game knowledge and PerGO usefulness and ease of use scores. This met the expectation that people who know (the complexity and variety of) pervasive games may consider PerGO to be more useful than others might. Also these people may find PerGO easier to understand and use. Differing from pervasive game knowledge, general computer game development experience did not obviously influence either PU or PEOU for both PerGO and GCCT. GCCT is designed to be applicable for general computer game creation, while PerGO is designed for a narrower domain: the pervasive games domain. From the above fact (pervasive game knowledge corresponds to higher PU while computer game knowledge does not), it can be inferred that it may not be worthwhile to develop a similar ontology for the general computer game domain. Whether it is worthwhile or not to develop similar ontologies for other specific game domains remains a question as well. For pervasive games, due to their novelty, large variety, and high complexity, developing a domain vocabulary like PerGO in order to more easily reuse domain knowledge can be a worthwhile endeavour. 

Computer game development experience does not correlate to PU for both PerGO and GCCT. This fact may indicate that PerGO and GCCT provide chances for amateurs to develop games easily, so that people with no game development experience could find the approach useful and easy to use in the same way as those who have such experiences. However, this does not exclude the possibility that having more specific game development experience might be correlated to how a person perceives the usefulness and ease of use. For example, if the distinction is made between the experience of using authoring tools and the experience of using more general game engines, a different result might be achieved. 

Specific model driven expertise does not hinder the acceptance of either PerGO or GCCT, as the results show in section 7.7.2. Participants who did not have any MDD knowledge perceived a similar ease of use for both PerGO and GCCT (based on the conceptual illustration in the videos). This indicates that with proper specialization and re-explaining, there is no extra difficulty in involving people without a MDD background in a GCCT process. Most traditional computer game developers do not have a MDD background. Therefore, given a proper working environment and support (in PerGO and GCCT), such developers might transition to using a GCCT approach with not much effort, in spite of a lack of MDD knowledge.
7.8.2 Threats to Validity

The most important threats to the validity of this evaluation are discussed. The concepts of construct validity, internal validity and external validity defined in [258] will be used as the basis for this discussion.

**Construct validity** is concerned with the degree to which inferences are warranted, from 1) the observed persons, settings, and cause and effect operations to 2) the constructs that these instances might represent. In other words, it is about whether the sampling particulars can be defended as measures of general constructs [258]. The goal is to evaluate the degree of acceptance and to explore potential impacts to the acceptance of PerGO and GCCT from the point of view of potential game developers. For the degree of acceptance, the TAM model was used as the basis on which to evaluate PU, PEOU, and IU for both PerGO and GCCT. TAM is a widely used model and the constructs have been rigorously validated. However, when the scales were developed for PU and PEOU, the usage scenario and subjects were adapted to fit the study. It was found that some of the scales might be more suitable for some utilitarian systems instead of vocabularies or approaches. After adaption, some scales may not work as well as the original ones. This may contribute to the uncertainty of the answers provided by respondents. Further, for the potential impacts, only found kinds of domain knowledge were investigated. As was reflected upon, having a more specific knowledge of game development may have had an impact. Other potential impacts are included, e.g. games domains, game budgets, and other available choices.

**Internal validity** is about “the validity of inferences about whether observed covariation between A (the presumed treatment) and B (the presumed outcome) reflects a causal relationship from A to B as those variables were manipulated or measured” [258]. In this study, Cronbach’s alpha was calculated to test internal consistency reliability. Correlation and regression were used to validate the TAM models. Although these statistical methods are effective in general, using them on small samples may, to some degree, threaten the validity. There were 46 respondents in total. This number is not large. However, it was more problematic to try to determine whether or not specific domain knowledge impacts how a person perceives PU and PEOU. For instance, according to the demographics of the respondents, only 2 people were experienced game developers, and only 3 people were very familiar with pervasive game concepts. Therefore the only comparison that could be made was between those with such knowledge and those without any such knowledge. The small size of the survey may also partly explain why PEOU was not a predictor IU for either PerGO or GCCT.

**External validity** is concerned with whether or not a causal relationship holds for variations 1) in persons, settings, treatments, and outcomes that were in the study and 2) for persons, settings, treatments, and outcomes that were not in the study [258]. This is about to what extent the study result can be generalized to other situations and other people. Most of the respondents in this survey were students with preliminary IT knowledge, like programming and modelling. In addition, most of the respondents did not have any real game development experiences in industry. Thus the study results may not apply to people who do not have a programming background, or real game developers in companies.
7.9 Summary
The main parts of PerGO and GCCT have been introduced and evaluated internally in Chapters 5 and 6. In this chapter, an empirical study was introduced regarding the potential user acceptance of PerGO and GCCT. A survey was performed to estimate how useful and easy to use PerGO and GCCT were from the point of view of potential users. Research was carried out to determine whether specific domain knowledge influences the degree of user acceptance. The results showed that, overall, both PerGO and GCCT were perceived to be useful, easy to use, and might be adopted by most of the respondents. The results also showed that people with some pervasive game knowledge might find PerGO to be more useful and more easy to use. However, knowledge of general computer game development and MDD expertise did not obviously influence the usefulness and the ease of use of PerGO or GCCT.
Chapter 8 Evaluation and Conclusion

In this chapter the results are assessed and the thesis concluded. The main contributions were introduced in Chapters 4-7.

Firstly, the quality of the research questions is evaluated. Then, the contributions are evaluated according to how they answered the research questions, and how they contribute to the state-of-the-art. Later, the main research method is evaluated according to the general guidelines introduced in Chapter 3. Lastly, future work is discussed, and this thesis concluded.

8.1 Evaluation of Research Questions

The research questions are evaluated according to some general criteria.

8.1.1 Criteria for Good Research Questions

According to [259], good research questions have some basic characteristics:

- **Clear**: They should be unambiguous and easy to understand.
- **Specific**: They should be sufficiently specific for it to be clear what constitutes an answer.
- **Answerable**: It should be possible to see what data are needed to answer them and how those data must be collected.
- **Interconnected**: The questions should be related in some meaningful way, forming a coherent whole.
- **Substantively relevant**: They should be worthwhile, non-trivial questions worthy of the research effort to be expended.

In order to evaluate the research questions, they are each classified according to the purposes of enquiry as either “exploratory”, “descriptive”, “explanatory”, or “emancipatory”. “The exploratory question is almost exclusively of flexible design. It is to find out what is happening, particularly in little-understood situations to seek new insights or to assess phenomena in a new light or to generate ideas and hypotheses for future research. The features of descriptive question are typically to portray an accurate profile of persons, events or situation and require extensive previous knowledge of the situation, to be researched or described, so that you know appropriate aspects on which to gather information. The explanatory question seeks an explanation of a situation or problem, traditionally but not necessarily in the form of relationships and explains patterns relating to the phenomenon being researched to identify relationships between aspects of the phenomenon. The features of emancipatory questions are to create opportunities and the will to engage in social action [259].”
8.1.2 Evaluation of Research Questions

The types of research questions are examined, and the five characteristics of each research question are evaluated according to the criteria introduced in section 8.1.1. Table 41 summarizes the evaluation results. The research questions are clear. According to the audience (working partner, supervisors, and reviewers) who have read this thesis, the research questions were understood in the same way without special issues being raised. The questions are specific as what constitutes the answers is clear (see Table 41 for details). These questions are also answerable. The research methods used towards the contributions met the original expectations of the research questions (details in Table 41). Further, the research questions are inter-connected and relevant to the challenges derived from the state-of-the-art (as introduced in Chapter 3). Figure 96 illustrates the relationship between research questions and how they answer the challenges.

- **RQ1**: What important concepts need to be considered regarding creating pervasive games with a model driven approach?
  - RQ1.1: What important characteristics should/may a pervasive game have?
  - RQ1.2: What concepts can be used in a Domain Specific Language (DSL) of pervasive games?
- **RQ2**: How can MDSD techniques be applied in a traditional pervasive/computer game creation process?
  - RQ2.1: How can a formalized domain vocabulary be used to enhance the domain analysis process in order to create pervasive games with a DSM approach?
  - RQ2.2: How can a traditional computer game development process be adapted to support DSM tasks in an efficient and iterative way?

![Figure 96. Research Questions and Challenges](image-url)
Table 41. Evaluation of Research Questions

<table>
<thead>
<tr>
<th>RQ</th>
<th>Clear</th>
<th>Specific</th>
<th>Type</th>
<th>Answerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1.1</td>
<td>More than three persons understood them in the same way, no ambiguity</td>
<td>A conceptual framework (for features) constitutes the answer</td>
<td>Descriptive</td>
<td>Concepts for important pervasiveness characteristics can be collected, analysed and classified by literature review</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ1.2</td>
<td></td>
<td>A pervasive domain specific ontology (with concepts for DSL use) constitutes the answer</td>
<td>Descriptive</td>
<td>Lower level concepts can be built upon general computer game theories. Higher level concepts can be derived from lower level concepts to fulfill pervasive game features</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ2.1</td>
<td></td>
<td>A regulated procedure (stepwise) based on a pre-defined domain vocabulary constitutes the answer</td>
<td>Exploratory</td>
<td>The steps can be derived from a general domain analysis procedure and are based upon the domain vocabulary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ2.2</td>
<td></td>
<td>An adapted MDD procedure (stepwise) for the computer game domain constitutes the answer</td>
<td>Exploratory</td>
<td>The procedure should be a specialization of a general MDD procedure which considers computer game development conventions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.2 Evaluation of Contributions

The research contributions are discussed according to how they answered the research questions and how they contributed to the state-of-the-art.

8.2.1 Contributions and Research Questions

The three contributions answered the four research questions raised previously. RQ1.1, what important characteristics should/may a pervasive game have, was answered by the first research contribution: the TeMPS framework (as introduced in Chapter 4). In TeMPS, four perspectives are defined: temporality, mobility, perceptibility and sociality. Each perspective includes several aspects. For instance, perceptibility includes how players could control the game, by utilizing their location, gesture, routine information, etc. These aspects characterize how a game can be called “pervasive”. According to [20], pervasiveness occurs in different perspectives and levels of intensity, and we need to be aware of the context and the perspectives, while let go of “the notion of coining the ultimate definition of pervasive games”. Thus, instead of
defining “what is a pervasive game”, TeMPS tries to answer “why the game is pervasive” and “how pervasive the game is” by structuring the possible characteristics that a pervasive game should/may have. A review of approximately 30 pervasive games showed that TeMPS fulfilled this requirement.

RQ1.2, what concepts can be used in a Domain Specific Language (DSL) of pervasive games, was answered by the contribution of PerGO, as illustrated in Chapter 5. PerGO defines two levels of concepts representing constructs within the orthogonal dimensions of a game such as gameplay, AI, elements, control and presentation. As the higher level concepts are defined and structured based on common computer game terminologies, the lower level concepts are derived from higher level concepts aiming to capture/ implement pervasive game specific features. All concepts are evaluated according to whether they are of proper abstraction and complexity level, and whether they are appropriate to be used in a DSM environment. In addition, PerGO also answers RQ2.1, how can a formalized domain vocabulary be used to enhance the domain analysis process in order to create pervasive games with a DSM approach. A domain analysis procedure was defined (see section 5.3). The procedure was illustrated based on PerGO (as a formalized vocabulary for the pervasive game domain). It also works with other domain vocabularies which have similar structures. Its usage was illustrated in one example and two case studies. The results showed that it helped to structure the domain analysis and improved the overall DSM efficiency. User acceptance testing, presented in Chapter 7, showed that concepts in PerGO are easy to understand and the vocabulary was perceived as useful for domain analysis, for both domain experts and amateurs. The way that the formalized domain vocabulary (PerGO) is used to enhance the domain analysis process was thought of as useful.

The contribution of GCCT answers RQ2.2, how can a traditional computer game development process be adapted to support DSM tasks in an efficient and iterative way. GCCT can be thought of as a specialized DSM process for the computer game domain. By reusing existing game creation tasks and automating general tasks, GCCT illustrates a compact and practical DSM approach which respects game creation traditions. In GCCT, traditional computer game development tasks like game design and level design are adapted to provide inputs for the DSM definition. On the other hand, DSM editors and code generators could enhance level designing and game code automation. The overall GCCT approach is still iterative, which facilitates both the game design evolution and the development of DSM tools. Two case studies were performed, as well as one user acceptance survey. The results (as shown in Chapters 6 and 7) were also positive.

8.2.2 Contributions and State-of-the-Art

The TeMPS framework was constructed to better describe pervasive game features. As mentioned in section 3.1, previous researchers have indicated that there is no commonly accepted definition for pervasive games (C2). In addition, the term “pervasive game” has often had different meanings due to different contexts and perspectives. Therefore, it was of interest to see how games can be pervasive by identifying the typical/ novel features they have, compared with those of traditional computer games. When approximately 30 pervasive games were analysed and classified, it was found that TeMPS highlighted the pervasiveness features in a structured way. As a
result, TeMPS clarifies what should be considered when designing a pervasive game, or when designing features to “pervasify” a traditional computer game.

While TeMPS describes high level features to ease pervasive game design, PerGO contributes to a structured collection of constructive blocks that can be used to actually build pervasive games. Apparently, PerGO is connected with TeMPS as the constructive blocks should be able to be integrated to implement the features described in TeMPS. On the other hand, PerGO is based on lower level concepts when considering the feasibility of MDD application. As pointed out by previous researchers, there is no common vocabulary for this aim, neither in the computer game domain, nor in the pervasive game domain (C1). PerGO provides two levels of concepts. The higher level concepts are applicable for general computer games, while the lower level concepts are derived from higher level ones for pervasive games. In this way, for other specific computer game domains, it is also possible to build a similar ontology based on the higher level concepts. Thus PerGO can be considered as providing a common domain vocabulary structure for computer games, as well as a specialized domain vocabulary for pervasive games.

As reviewed in Chapter 2, the state-of-the-art application of MDD in the game domain is quite new and immature, from many perspectives. Considering this situation, two issues were highlighted, as mentioned in section 3.1: 1) domain analysis, as an important task to provide qualified data for further MDD tasks, was not carried out in a rigorous way (C3); 2) the overall procedure was not designed carefully to fit the requirements of both the MDD part and the game development part (C4). In this research, a set of procedural steps were defined to perform domain analysis based on the PerGO domain vocabulary. As was evaluated in Chapters 5 and 7, PerGO is effective and accepted by potential users. PerGO can be thought of as both a domain vocabulary (the ontology) and a domain analysis method (the procedure) based on the domain vocabulary for the pervasive game domain (for C3).

GCCT is designed to enhance the MDD procedure applied in the game domain (for C4). As mentioned in Chapter 2, the MDD procedure has many variables and involves many tasks. On the other hand, computer game development is very complex and has many peculiarities. For instance, game design is tightly connected with software development, and the two evolve together. With careful investigation, the potential was found for a more effective MDD approach for game domains. By adapting game tasks and MDD tasks to facilitate each other, and utilizing language work bench tools to streamline and automate tasks, the GCCT approach was defined as an instantiation of MDD in the game domain. Case studies were performed (see Chapter 6), and a user acceptance survey was conducted (see Chapter 7). The results showed that GCCT worked effectively and efficiently. In addition, the overall approach was welcomed by potential users.

8.3 Evaluation of Design Science

The design science paradigm was used for this research, as introduced in Chapter 3. In this section, the methods are evaluated and reflected upon according to the design science guidelines proposed in [23, 176].

Guideline 1: Design as an artefact: one or more artefacts must be produced by design science research. The artefact can be a construct, a model, a method, or an instantia-
tion. In this thesis, three research results were introduced. The TeMPS framework is a conceptual model. PerGO consists of a conceptual model (ontology part) and a method (domain analysis procedure part), while GCCT is a method to apply MDD in the game domain.

**Guideline 2: Problem Relevance:** the design science research should develop technology-based solutions to address important and relevant business problems. Prior to constructing the artefacts, a thorough review was performed in both the game domain and MDD domain, as well as the MDGD domain. The challenges of applying MDD in the game domain were identified, which were not addressed in the current MDD applications for games development. As discussed in Chapter 2, in the MDGD domain, domain analysis and overall procedural design were not addressed in a serious manner despite their potentially crucial impacts to the ultimate effectiveness and efficiency of MDD application. These issues partly stemmed from a lack of common vocabularies available for MDD usage in the game domain. Furthermore, the pervasive games domain even lacks a common definition for pervasive games or pervasiveness features. The contributions of this thesis were made to address these important issues, as stated in section 8.2.

**Guideline 3: Design Evaluation:** well-executed evaluation methods must be used to rigorously demonstrate the utility, quality and efficacy of design artefacts. For TeMPS, its usage was demonstrated by analysing approximately thirty pervasive games, and identifying their pervasiveness features according to the framework. The results show that the four perspectives are orthogonal and the overall framework can help analyse and classify pervasive games in an effective and efficient manner. For PerGO, its usage was demonstrated by an example (see Chapter 5) and two case studies (see Chapter 6). Its effectiveness was evaluated as a vocabulary and efficiency as a domain analysis process based on internal discussion and analysis. In addition, a survey was performed to evaluate how useful and easy to use PerGO is from the perspective of potential users. Case studies and a survey were used for the evaluation of GCCT as well. In addition to the productivity data collected, the cost data was structured in order to aim for a more objective analysis of a MDD process like GCCT. GCCT’s usage was demonstrated in the RealCoins case study, data presented, and the efficiency of GCCT analysed. Finally, the acceptance of GCCT by potential game developers was investigated by the user survey. The results showed that, overall, GCCT was easy to understand and might be adopted by the majority of the participants.

**Guideline 4: Research Contribution:** clear and verifiable contributions in the areas of the design artefact, design foundations, and/or design methodology. In this thesis, three contributions were presented which are design artefacts. All of them were formally defined. The TeMPS framework consists of four orthogonal perspectives and aspects within them. All the perspectives and concepts are defined and illustrated with samples (see Chapter 5). PerGO consists of two levels of concepts describing six perspectives of computer games. The levels, perspectives, as well as the main concepts, are defined and illustrated with figures and samples (see Chapter 5). For GCCT, the four steps were also formally defined and it was illustrated how GCCT was derived from general MDD approaches and how it connects to game tasks (see Chapter 6). The important metrics of all contributions were verified, both theoretically and practically.
Guideline 5 Research Rigor: Rigorous methods should be applied in both the construction and evaluation of the design artefact. According to [23], rigor is achieved by using existing knowledge in design science. For TeMPS, the framework was constructed based on the existing discourse regarding the definition and genres of pervasive games. TeMPS was evaluated by analysing and classifying pervasive features of around thirty pervasive games. For PerGO, the lower level (computer game part) structure and concepts were constructed based on generally accepted computer game theories and terminologies. In addition, the higher level concepts were derived based on the lower level parts, according to pervasive game specialities. The lower level concepts (pervasive game part) were connected to TeMPS to verify whether the concepts could be used to implement pervasive features. Similarly, for GCCT, the tasks and characteristics of both the MDD process and traditional computer game development process were investigated. In addition, GCCT was derived from a general MDD process by adapting game tasks and utilizing tools. Both PerGO and GCCT were evaluated based on the TAM model which has been used widely to predict user acceptance of technologies.

Guideline 6: Design as a search. All research artefacts were designed and constructed in an incremental way. In this study, TeMPS was constructed first. Then PerGO and GCCT were constructed and evaluated simultaneously as they connect to each other more tightly. Several iterations were performed for each artefact. In the earlier phase of TeMPS construction, the main focus was on literature studies. Literature on pervasive game theories and cases was reviewed. These pervasiveness features were classified, the basic TeMPS framework constructed and then refined by reviewing more pervasive games and trying to fit them in the framework. The work was demonstrated and discussed to gain inputs and comments during the overall process. Similarly, the construction of PerGO and GCCT also began from studying the literature. For PerGO, computer game theories were reviewed and analysed, addressing high level elements/dimensions/frameworks, as well as lower level game design and game software development constructs. For GCCT, the literature was reviewed regarding the process of both MDD and traditional game creation, analysing differences and potential connections. After that, PerGO and MDD were constructed, and they were evaluated both internally and externally. Several iterations were performed. In the earlier stage, examples conceptually based on PerGO and MDD were analysed, and various MDD environments were practically tried out to gain direct experiences and lessons. Changes and enhancement were done continuously during this stage. Later, when PerGO and MDD were more consolidated, case studies were more formally performed to collect cost and productivity data. Lastly, a survey was used to ascertain acceptance from the point of view of potential users.

Guideline 7: Communication of Research. The research must be presented effectively to both a technology-oriented and management-oriented audience. The research presented in this thesis was presented to both audiences. This was ensured by discussing and cooperating with students and other colleagues, gaining comments from supervisors, receiving inputs from peer reviewers of conferences and journals, participating in and presenting at conferences, etc. PerGO and GCCT were also presented to a larger audience with two videos, and input was solicited regarding the usefulness and ease of use of them. However, due to the limitation of time and resources, more partici-
pants were not involved from the game industry, and they were not allowed to try out the approaches in practice. This will be improved upon in the future.

8.4 Future Work and Conclusion

In this thesis, the research results were presented regarding conceptual frameworks, and practical solutions given towards enhanced model driven techniques applied in the game domain, especially the pervasive game domain. These results answered the research questions and contributed to the state-of-the-art. Reflections also raised some opportunities for future research. Some possible future directions were outlined concerning the three artefacts proposed in this thesis.

The TeMPS framework resulted from the earlier research phase. Despite its ability to identify and classify pervasiveness features, there are some limitations and areas to improve on. Firstly, the four dimensions defined in TeMPS are quite uneven. The perception dimension contains several more aspects than the temporality and mobility dimensions. Secondly, the suggested set of perspectives or their options/aspects is not exhaustive, and can be refined later. For instance, Sociality plays an important role, and an important aspect is how players perceive each other’s existence and how they communicate (face-to-face real-time video and audio, versus pure text mails) [215]. Analysis in this area could help extend the TeMPS framework in the future. Thirdly, the TeMPS framework lacks the ability to describe the degree of pervasiveness and it does not specify which features are mandatory or optional for a pervasive game. Therefore, a better structured feature model may probably be designed in the future. Lastly, it was also realized that this framework is somewhat empirical and lacks solid theoretical ground. Efforts will be made to improve this in the future. In addition, the scoring criteria based on TeMPS can be made less subjective.

The current ontology of PerGO has not been refined and evaluated sufficiently with enough case studies in a complete game software development process. Thus directions have been identified for possible future works: First, PerGO will be extended. The basic structure of PerGO was designed for the general model driven usage of all computer games. Domain specific concepts may come from more channels than currently indicated by the current six perspectives. PerGO can be expanded to cover more perspectives such as sociality, network communication, architecture, etc. Second, the concepts can be refined by carrying out more case studies. It is not possible that one DSL/DSM can be designed once and used forever, and “a large part of its power comes from its ability to evolve" [17]. The ontology needs to be refined and evolved by more case studies.

For GCCT, the ultimate goal is to improve the overall process of model driven game development. In addition to the domain analysis and the procedural aspects which have currently been covered, more aspects of game creation may be explored to achieve the best results of applying MDD in the game domain. Aspects such as management, document transitions, and participants’ cooperation can also be covered by an extension of the current GCCT approach. Research questions of interest can include: how to utilize domain analysis results to produce DSM artefacts, how traditional game creation participants adapt themselves to new tasks, and how traditional documents used in the game creation process can be reused in the new process.
In addition to expanding and refining current PerGO and GCCT, their evaluation is also important. Prior to that, the practical environment and tools to support the usage of PerGO and GCCT should be constructed and enhanced. Therefore, more formal and rigorous evaluations should be carried out (especially on the improvement of the productivity), and the artefacts improved in iterations. In addition, more external inputs might be preferred for the evaluation. Larger scale user experiments and surveys can be performed to allow external participants to try out the approach and provide comments. More participants from the game industry might be recruited.
References


J. Juul, "The game, the player, the world: Looking for a heart of gameness," 2003, p. 121.


[203] A. D. Cheok, K. H. Goh, W. Liu, F. Farbiz, S. W. Fong, S. L. Teo, et al., "Human Pacman: a mobile, wide-area entertainment system based on physical,


Appendix A Survey Instrument

Creating Games with Customized Tools
User Acceptance Survey

We introduced an approach named GCCT (Game Creation with Customized Tools). With GCCT, we can develop tools, and then develop games with these tools. To develop the tools, we should clearly define the game features that will be supported by the tools. PerGO (Pervasive Game Ontology) is a pre-defined vocabulary to ease this work for the pervasive game domain.

Please use a few minutes to answer the following questions regarding them. Thank you for your participation!

**Personal Information**

1. Gender:  □ Male  □ Female
2. Age:      □ <20  □ 20-29  □ 30-39  □ 40-49
             □ 50-59  □ >59
3. Education: □ Lower  □ Bachelor  □ Master
             □ Ph.D.  □ Higher

<table>
<thead>
<tr>
<th>Question</th>
<th>Very Familiar</th>
<th>Know something</th>
<th>Know nothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. How familiar are you with Computer Game Development?</td>
<td>□  □  □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. How familiar are you with Pervasive Game Concepts/Theories?</td>
<td>□  □  □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. How familiar are you with UML Class Diagram?</td>
<td>□  □  □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. How familiar are you with DSL/DSM/MDSD?</td>
<td>□  □  □</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8. Do you plan to develop computer games in the future? □ Yes  □ Probably  □ No

**PerGO**

<table>
<thead>
<tr>
<th>Question</th>
<th>Strongly disagree</th>
<th>2</th>
<th>3</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Using PerGO would enable me to specify features of pervasive games more quickly.</td>
<td>□  □  □  □  □</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Using PerGO would improve my job performance of specifying features of pervasive games.</td>
<td>□  □  □  □  □</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Using PerGO to specify features of pervasive games would increase my productivity.</td>
<td>□  □  □  □  □</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Using PerGO would enhance my effectiveness on specifying features of pervasive games.</td>
<td>□  □  □  □  □</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Using PerGO would make it easier to specify features of pervasive games.</td>
<td>□  □  □  □  □</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. I find PerGO useful to specify pervasive game features.</td>
<td>□  □  □  □  □</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7. Learning to use PerGO would be easy for me.  
8. I find it easy to use PerGO to specify pervasive game features that I want.  
9. I find the perspectives and concepts of PerGO clear and understandable.  
10. I find PerGO to be flexible to use.  
11. It would be easy for me to become skillful at using PerGO.  
12. I find PerGO easy to use.  
13. Assuming I need to create pervasive games and I have access to the tools, I intend to use PerGO.  
14. Given that I need to create pervasive games and I have access to the tools, I predict that I would use PerGO.

---

**GCCT**

1. Using GCCT would enable me to create series of pervasive games more quickly.  
2. Using GCCT would improve my performance to create series of pervasive games.  
3. Using GCCT to create series of pervasive games would increase my productivity.  
4. Using GCCT would enhance my effectiveness on creating series of pervasive games.  
5. Using GCCT would make it easier to create series of pervasive games.  
6. I find GCCT useful to create series of pervasive games.  
7. Learning to use GCCT would be easy for me.  
8. I find it easy to use GCCT to create pervasive games that I want to create.  
9. I find the approach of GCCT clear and understandable.  
10. I find GCCT to be flexible to use.  
11. It would be easy for me to become skillful at using GCCT.  
12. I find GCCT easy to use.  
13. Assuming I need to create pervasive games and I have access to the tools, I intend to use GCCT.  
14. Given that I need to create pervasive games and I have access to the tools, I predict that I would use GCCT.

---

**Further Contact (Optional)**

1. Your Name:  
2. If you would accept further information/query/discussion
through email address? (Please leave your email address if yes)

3. If you would accept further query or discussion through phone? (please leave your phone number here if yes)

4. We appreciate that if you have any comments or suggestions:
### Appendix B Additional Descriptive Results

#### Table 42. Additional Descriptive Results for PerGO

<table>
<thead>
<tr>
<th>N=46</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Medium</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU1_1</td>
<td>3</td>
<td>5</td>
<td>4.0435</td>
<td>4</td>
<td>.72897</td>
</tr>
<tr>
<td>PU1_2</td>
<td>2</td>
<td>5</td>
<td>3.7609</td>
<td>4</td>
<td>.79400</td>
</tr>
<tr>
<td>PU1_3</td>
<td>2</td>
<td>5</td>
<td>3.7826</td>
<td>4</td>
<td>.84098</td>
</tr>
<tr>
<td>PU1_4</td>
<td>2</td>
<td>5</td>
<td>3.8261</td>
<td>4</td>
<td>.76896</td>
</tr>
<tr>
<td>PU1_5</td>
<td>2</td>
<td>5</td>
<td>3.9783</td>
<td>4</td>
<td>.80247</td>
</tr>
<tr>
<td>PU1_6</td>
<td>3</td>
<td>5</td>
<td>3.9565</td>
<td>4</td>
<td>.75884</td>
</tr>
<tr>
<td>PEOU1_1</td>
<td>2</td>
<td>5</td>
<td>3.3913</td>
<td>3</td>
<td>.77397</td>
</tr>
<tr>
<td>PEOU1_2</td>
<td>2</td>
<td>5</td>
<td>3.4130</td>
<td>3</td>
<td>.80488</td>
</tr>
<tr>
<td>PEOU1_3</td>
<td>1</td>
<td>5</td>
<td>3.3913</td>
<td>3</td>
<td>.90623</td>
</tr>
<tr>
<td>PEOU1_4</td>
<td>2</td>
<td>5</td>
<td>3.5217</td>
<td>3</td>
<td>.83637</td>
</tr>
<tr>
<td>PEOU1_5</td>
<td>2</td>
<td>5</td>
<td>3.5652</td>
<td>3</td>
<td>.86029</td>
</tr>
<tr>
<td>PEOU1_6</td>
<td>2</td>
<td>5</td>
<td>3.3913</td>
<td>3</td>
<td>.88137</td>
</tr>
<tr>
<td>IU1_1</td>
<td>2</td>
<td>5</td>
<td>3.7609</td>
<td>4</td>
<td>.82151</td>
</tr>
<tr>
<td>IU1_2</td>
<td>2</td>
<td>5</td>
<td>3.7391</td>
<td>4</td>
<td>.92939</td>
</tr>
</tbody>
</table>

#### Table 43. Additional Descriptive Results for GCCT

<table>
<thead>
<tr>
<th>N=46</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Medium</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU2_1</td>
<td>3</td>
<td>5</td>
<td>3.8913</td>
<td>4</td>
<td>.82269</td>
</tr>
<tr>
<td>PU2_2</td>
<td>3</td>
<td>5</td>
<td>3.7609</td>
<td>4</td>
<td>.70505</td>
</tr>
<tr>
<td>PU2_3</td>
<td>3</td>
<td>5</td>
<td>3.7826</td>
<td>4</td>
<td>.78636</td>
</tr>
<tr>
<td>PU2_4</td>
<td>3</td>
<td>5</td>
<td>3.6739</td>
<td>4</td>
<td>.73195</td>
</tr>
<tr>
<td>PU2_5</td>
<td>2</td>
<td>5</td>
<td>3.8043</td>
<td>4</td>
<td>.77802</td>
</tr>
<tr>
<td>PU2_6</td>
<td>3</td>
<td>5</td>
<td>3.8261</td>
<td>4</td>
<td>.79734</td>
</tr>
<tr>
<td>PEOU2_1</td>
<td>2</td>
<td>5</td>
<td>3.3696</td>
<td>3</td>
<td>.77053</td>
</tr>
<tr>
<td>PEOU2_2</td>
<td>2</td>
<td>5</td>
<td>3.3696</td>
<td>3</td>
<td>.74113</td>
</tr>
<tr>
<td>PEOU2_3</td>
<td>2</td>
<td>5</td>
<td>3.6087</td>
<td>4</td>
<td>.97703</td>
</tr>
<tr>
<td>PEOU2_4</td>
<td>2</td>
<td>5</td>
<td>3.5870</td>
<td>3</td>
<td>.74762</td>
</tr>
<tr>
<td>PEOU2_5</td>
<td>2</td>
<td>5</td>
<td>3.4130</td>
<td>3</td>
<td>.77678</td>
</tr>
<tr>
<td>PEOU2_6</td>
<td>2</td>
<td>5</td>
<td>3.3478</td>
<td>3</td>
<td>.76645</td>
</tr>
<tr>
<td>IU2_1</td>
<td>2</td>
<td>5</td>
<td>3.7174</td>
<td>4</td>
<td>.88602</td>
</tr>
<tr>
<td>IU2_2</td>
<td>2</td>
<td>5</td>
<td>3.7826</td>
<td>4</td>
<td>.96409</td>
</tr>
</tbody>
</table>
Appendix C Additional TAM Results for PerGO

Table 44. Regression for PU1, PEOU1 and IU1

<table>
<thead>
<tr>
<th>Model</th>
<th>R</th>
<th>R2</th>
<th>Adjusted R2</th>
<th>Std. Error of the Estimate</th>
<th>Change Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td>R2 Change</td>
<td>F Change</td>
</tr>
<tr>
<td>1</td>
<td>.791*</td>
<td>.626</td>
<td>.609</td>
<td>.52635</td>
<td>.626</td>
<td>36.027</td>
</tr>
</tbody>
</table>

a. Predictors: (Constant), PEOU1, PU1.

Table 45. Coefficients for PU1, PEOU1 and IU1

<table>
<thead>
<tr>
<th>Model</th>
<th>Unstandardized Coefficients</th>
<th>Standardized Coefficients</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>Std. Error</td>
<td>Beta</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(Constant)</td>
<td>-.139</td>
<td>.471</td>
<td>-.294</td>
</tr>
<tr>
<td></td>
<td>PU1</td>
<td>.719</td>
<td>.226</td>
<td>.562</td>
</tr>
<tr>
<td></td>
<td>PEOU1</td>
<td>.317</td>
<td>.218</td>
<td>.257</td>
</tr>
</tbody>
</table>

a. Dependent Variable: IU1

Table 46. Regression for PEOU1 and PU1

<table>
<thead>
<tr>
<th>Model</th>
<th>R</th>
<th>R2</th>
<th>Adjusted R2</th>
<th>Std. Error of the Estimate</th>
<th>Change Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td>R2 Change</td>
<td>F Change</td>
</tr>
<tr>
<td>1</td>
<td>.849*</td>
<td>.721</td>
<td>.715</td>
<td>.35110</td>
<td>.721</td>
<td>113.83</td>
</tr>
</tbody>
</table>

a. Predictors: (Constant), PEOU1.

Table 47. Coefficients for PEOU1 and PU1

<table>
<thead>
<tr>
<th>Model</th>
<th>Unstandardized Coefficients</th>
<th>Standardized Coefficients</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>Std. Error</td>
<td>Beta</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(Constant)</td>
<td>1.071</td>
<td>.269</td>
<td>3.977</td>
</tr>
<tr>
<td></td>
<td>PEOU1</td>
<td>.818</td>
<td>.077</td>
<td>.849</td>
</tr>
</tbody>
</table>

a. Dependent Variable: PU1
# Appendix D Additional TAM Results for GCCT

## Table 48. Regression for PU2, PEOU2 and IU2

<table>
<thead>
<tr>
<th>Model</th>
<th>R</th>
<th>R2</th>
<th>Adjusted R2</th>
<th>Std. Error of the Estimate</th>
<th>Change Statistics</th>
<th>Sig. F Change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R2 Change</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F Change</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>df1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>df2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>.834a</td>
<td>.696</td>
<td>.682</td>
<td>.51065</td>
<td>.696</td>
<td>49.206</td>
</tr>
</tbody>
</table>

a. Predictors: (Constant), PU2, PEOU2.

## Table 49. Coefficients for PU2, PEOU2 and IU2

<table>
<thead>
<tr>
<th>Model</th>
<th>Unstandardized Coefficients</th>
<th>Standardized Coefficients</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>Std. Error</td>
<td>Beta</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(Constant)</td>
<td>-.553</td>
<td>.445</td>
<td>-1.244</td>
</tr>
<tr>
<td></td>
<td>PU2</td>
<td>.776</td>
<td>.224</td>
<td>.565</td>
</tr>
<tr>
<td></td>
<td>PEOU2</td>
<td>.395</td>
<td>.216</td>
<td>.298</td>
</tr>
</tbody>
</table>

a. Dependent Variable: IU2

## Table 50. Regression for PEOU2 and PU2

<table>
<thead>
<tr>
<th>Model</th>
<th>R</th>
<th>R2</th>
<th>Adjusted R2</th>
<th>Std. Error of the Estimate</th>
<th>Change Statistics</th>
<th>Sig. F Change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R2 Change</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F Change</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>df1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>df2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>.857a</td>
<td>.734</td>
<td>.728</td>
<td>.34366</td>
<td>.734</td>
<td>121.32</td>
</tr>
</tbody>
</table>

a. Predictors: (Constant), PEOU2.

## Table 51. Coefficients for PEOU2 and PU2

<table>
<thead>
<tr>
<th>Model</th>
<th>Unstandardized Coefficients</th>
<th>Standardized Coefficients</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>Std. Error</td>
<td>Beta</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(Constant)</td>
<td>.940</td>
<td>.264</td>
<td>3.567</td>
</tr>
<tr>
<td></td>
<td>PEOU2</td>
<td>.826</td>
<td>.075</td>
<td>.857</td>
</tr>
</tbody>
</table>

a. Dependent Variable: PU2