Title Gamification as a Service - Conceptualization of a Generic Enterprise Gamification Platform

Submitted by Philipp Herzig, M.Sc.
Born on February 23, 1987 in Zittau

in partial fulfillment of the requirements for the degree

DOKTOR-INGENIEUR

(Dr.-Ing.)

Primary reviewer Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill, Technische Universität Dresden
Secondary reviewer Prof. Dr. rer. pol. habil. Rüdiger Zarnekow, Technische Universität Berlin
Defended on July 2, 2014

Dresden, July 9, 2014
# CONTENTS

<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>IX</td>
</tr>
<tr>
<td>List of Tables</td>
<td>XIII</td>
</tr>
<tr>
<td>List of Listings</td>
<td>XV</td>
</tr>
<tr>
<td>List of Abbreviations</td>
<td>XVII</td>
</tr>
<tr>
<td>Abstract</td>
<td>XXI</td>
</tr>
<tr>
<td>Acknowledgement</td>
<td>XXIII</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Motivation .................................................................................. 1
1.2 Research Objective and Research Questions .................................. 2
1.3 Research Design .......................................................................... 2
1.4 Thesis Structure ......................................................................... 2
1.5 Publications ................................................................................ 4
1.5.1 Conference and Journal Papers ................................................ 4
1.5.2 Patents ..................................................................................... 5

## 2 Foundations

2.1 Gamification ................................................................................ 7
2.1.1 Definition ................................................................................ 7
2.1.2 Emergence and Progression Aspects of Games ............................. 10
2.1.3 Biology & Psychology of Games ............................................... 10
2.2 Enterprise Architectures ............................................................ 12
2.2.1 Overview .................................................................................. 12
2.2.2 Service Oriented Architecture ............................................... 13
2.2.3 Event Driven Architecture ...................................................... 16
2.2.3.1 Definition, Characteristics, and Building Blocks .................. 16
2.2.3.2 Complex Event and Stream Processing .................................. 17
2.2.4 Production Rules ...................................................................... 20
Contents

D.2 Graphical Artifacts of Gamification Platform ........................................... 236
D.3 RETE Graph of Networking Lunch Rules ................................................. 239

E Statistical Tests .................................................................................... 241
E.1 (Multivariate) Analysis of Variance ......................................................... 241
E.2 Wilcoxon-Mann-Whitney Test ................................................................. 242
E.3 Jarque-Bera Test .................................................................................... 243
E.4 t-Test ................................................................................................. 243
## LIST OF FIGURES

1.1 Overall structure of this thesis .............................................. 3

2.1 Classification of gamification ............................................. 8

2.2 An exemplary logical service-oriented architecture .................. 14

2.3 Components of an event-driven architecture .......................... 17

2.4 Abstraction of events ..................................................... 19

2.5 Exemplary temporal operators ........................................... 19

2.6 Example instance of a RETE graph ..................................... 21

2.7 Entity-based state management for complex event processing .... 23

3.1 Workflows in the Rational Unified Process ............................ 28

3.2 Adapted gamification development process ............................ 31

4.1 Constituting elements of mission graphs ............................... 38

4.2 Example instance of mission graph model ............................ 39

5.1 Definition of Space element .............................................. 52

5.2 Definition of Game Level element ...................................... 53

5.3 General condition syntax ................................................ 55

5.4 Boolean constraint and conditions syntax .............................. 57

5.5 Team condition syntax .................................................... 58

5.6 General consequence element ............................................ 59

6.1 Example for Machinations framework ................................... 67

6.2 Complex graphical game model in Machinations framework ....... 68

6.3 High-level architecture of an achievement system .................. 71

6.4 High-level architecture of an integrated solution .................... 73

6.5 Action definition example from the Gigya solution ................. 74

6.6 High-level architecture of a generic gamification platform ....... 76

7.1 General scenario for the technical introduction of gamification into enterprise information systems ......................... 83

7.2 Stateful complex event processing based on business entities .... 86

7.3 Synchronous context update procedure ................................ 90

7.4 Asynchronous context update procedure .............................. 91

7.5 Player and avatar perspective of data model .......................... 93
List of Figures

7.6 Enumerations used in the data model ........................................ 93
7.7 Procedure for processing received events within the EPA .................. 96
7.8 Conceptual transformation of GaML into code for the gamification runtime environment .................................................. 101
7.9 Summary of the conceptual translation procedure ............................. 115
8.1 Example screenshot from the SAP TwoGo gamification project .............. 118
8.2 Example screenshots from the SAP Networking Lunch gamification project119
8.3 Example screenshots from the SAP Financial Fact Sheet gamification project .......................................................... 120
8.4 Example screenshot from the Soccer application with gamification ......... 121
8.5 Example figures from the CHIO gamification project .......................... 122
8.6 Carpooling example in GaML editor ........................................... 124
8.7 OCL expressions applied in Ecore meta-model ................................ 125
8.8 Compile process of GaML into code for the gamification runtime environ- 126
8.9 Example of code generation within GaML editor .............................. 126
8.10 Components of gamification runtime environment ........................... 127
8.11 Component diagram of AchievementProvider .................................. 128
8.12 Component diagram of RuleEngine .............................................. 129
8.13 Component diagram of Gateway ................................................ 130
8.14 Examples for the administration user interface ................................ 131
8.15 Example instance of notification widget ........................................ 132
8.16 Example instance of mission widget ............................................. 132
8.17 Assemble of different gamification widgets ..................................... 133
8.18 Integrated gamification widgets within mobile application ................. 134
8.19 Component diagram of gamification platform in synchronous mode ..... 135
8.20 Component diagram of gamification platform in asynchronous mode .... 136
8.21 Integration architecture for SAP TwoGo application .......................... 139
8.22 Integration architecture for SAP Networking Lunch application .......... 142
8.23 Measurement points for synchronous model .................................. 144
8.24 Measurement points for asynchronous model ................................ 145
8.25 Experimental average response times ......................................... 146
8.26 Experimental maximum response times ....................................... 147
8.27 Comparison of response time distributions between synchronous and asynchronous case at 1024 experimental users ....................... 147
8.28 Evolution of response times for synchronous communication ............. 148
8.29 Slow response times compared to a fixed 500ms threshold ................. 149
8.30 Space-time trade-off of response-time versus inverse memory consump- 149
8.31 Distribution of event interarrival times for 250 users and 0.5 events per tion in the event processing agent ................................................ 149
user-second .............................................................................. 152
8.32 Response time prediction for synchronous case ............................... 156
8.33 Response time prediction for asynchronous case .............................. 156
8.34 System usability scale item averages .......................................... 162
8.35 Variable means with standard error of ungamified (A) versus gamified (B) SAP Financial Fact Sheet application ............................. 165
8.36 Integration architecture for SAP Financial Fact Sheet application ........ 168
A.1 Terminal rules .......................................................................... 195
A.2 Terminal rules 2 ...................................................................... 196
| A.3 | General numeric expressions | 197 |
| A.4 | Operand types | 198 |
| A.5 | Space syntax | 199 |
| A.6 | Game level syntax | 200 |
| A.7 | Event syntax | 201 |
| A.8 | Achievement syntax | 201 |
| A.9 | Point syntax | 202 |
| A.10 | Mission syntax | 203 |
| A.11 | Skill syntax | 204 |
| A.12 | Level syntax | 204 |
| A.13 | Good syntax | 205 |
| A.14 | Role syntax | 205 |
| A.15 | Leaderboard syntax | 206 |
| A.16 | General condition syntax | 207 |
| A.17 | Boolean constraints and conditions syntax for player | 208 |
| A.18 | Player condition syntax | 208 |
| A.19 | Boolean constraints and conditions syntax for teams | 209 |
| A.20 | Team condition syntax | 209 |
| A.21 | Event condition syntax | 210 |
| A.22 | Event condition right term syntax | 211 |
| A.23 | Condition references syntax | 211 |
| A.24 | Condition item references syntax | 212 |
| A.25 | Skill condition syntax | 212 |
| A.26 | Point condition syntax | 212 |
| A.27 | Random condition syntax | 212 |
| A.28 | Location condition syntax | 213 |
| A.29 | Leader condition syntax | 213 |
| A.30 | Role condition syntax | 213 |
| A.31 | Event condition syntax | 214 |
| A.32 | General consequence syntax | 215 |
| A.33 | Event consequence syntax | 216 |
| A.34 | Point consequence syntax | 216 |
| A.35 | Notification consequence syntax | 217 |
| A.36 | Narration consequence syntax | 217 |
| A.37 | General consequence syntax | 218 |
| C.1 | Correlation diagram for variables of GaML study | 233 |
| D.1 | Class diagram for associations with image entity | 235 |
| D.2 | Class diagram for event model | 236 |
| D.3 | Rule editor in the gamification platform prototype | 236 |
| D.4 | Gamification analytics in the gamification platform prototype | 237 |
| D.5 | Administration of user profile in the gamification platform prototype | 237 |
| D.6 | Additional widgets in the gamification platform prototype | 238 |
| D.7 | Optimized RETE graph representation for the SAP Networking Lunch application | 239 |
LIST OF TABLES

2.1 Levels of game design elements ........................................... 9
2.2 Comparison of intrinsic factors leading to work and game engagement . 11
2.3 General requirements for enterprise information systems ............... 13
2.4 Characteristics of services .................................................. 15
2.5 Comparison of characteristics in service-oriented and event-driven architectures .................................................. 24

3.1 Sequence of tasks in the design and specification workflow .......... 30

4.1 Visual game design elements ............................................... 40
4.2 Aggregated visual game design elements ................................ 43

5.1 Mapping of conceptual requirements and language specification .... 64

6.1 Comparison of existing game languages .................................. 70
6.2 Analysis of achievement systems ......................................... 72
6.3 Analysis of integrated gamification solutions .............................. 76
6.4 Analysis of generic gamification platforms ................................ 80
6.5 Comparison of system classes for gamification solutions ............... 82

7.1 Examples for gamification-related state information managed by the business entity provider ........................................ 87
7.2 Examples for gamification-related conditions managed by the event processing agent ........................................... 87
7.3 Examples for gamification-related consequences triggered by the event processing agent ........................................... 88
7.4 Mapping of rule types on architecture components ...................... 89
7.5 Reserved events in the gamification runtime environment and their pre-defined semantics ........................................... 100

8.1 Quantities of rule types in the SAP Networking Lunch application ...... 143
8.2 Summary of examined cost factors in the performance model ........... 151
8.3 Service time polynomials for number of users cost factor in synchronous mode ........................................... 153
8.4 Service time polynomials for number of users cost factor in asynchronous mode ........................................... 154

XIII
List of Tables

8.5  Quantitative validation of cost model comparing measured and predicted response times .................................................. 154
8.6  Validation of cost model comparing measured and predicted response times for SAP Networking Lunch application .......................... 155
8.7  Distribution of variable GEND .................................................. 160
8.8  Descriptive statistics for variables ........................................... 160
8.9  Correlation matrix of exogenous variables ................................. 161
8.10 A/B Analysis of ungamified (A) versus gamified (B) application ................................................................. 165
8.11 Time exposure for complete gamification enablement ................. 169

C.1  System usability scale questions .............................................. 232
LISTINGS

5.1 Constraint for uniqueness of IDs ............................................. 61
5.2 Constraints for point type semantics ........................................ 61
5.3 Constraint for point semantics .................................................. 61
5.4 Constraint and operations for acyclic missions ............................. 62
5.5 Constraint for type checks in numeric expressions .......................... 63

6.1 Excerpt of TicTacToe in game description language ....................... 66

7.1 Core rule for user-accepted missions (accept) ............................... 97
7.2 Core rule for user-accepted missions (decline) ............................. 97
7.3 Core rule for deleting point-in-time events ................................. 98
7.4 Core rule for deleting interval events ....................................... 98
7.5 Core rule for activating skills ................................................. 98
7.6 Core rule for new users or avatars ........................................... 99
7.7 Translation example for the point entity .................................. 102
7.8 Translation example for badge entity ...................................... 102
7.9 Translation example for badge entity including an optional when clause . 102
7.10 Rule assembled from Listing 7.9 ............................................. 103
7.11 Example translation of an outer player to avatar entity .................. 103
7.12 Example translation of an outer team entity ................................ 103
7.13 Simple example for variable binding ...................................... 104
7.14 Complex example for variable binding .................................... 104
7.15 Example translation of Boolean operators .................................. 105
7.16 Example translation of general reference conditions ....................... 105
7.17 Example translation of several mission conditions ........................ 106
7.18 Example translation of skill condition ..................................... 106
7.19 Example translation of point condition .................................... 106
7.20 Example translation of luck condition .................................... 107
7.21 Example translation for location condition ................................ 107
7.22 Example translation of leader condition ................................... 107
7.23 Example translation of role condition .................................... 108
7.24 Example translation for player-team relationships ........................ 108
7.25 Translation of event type ..................................................... 109
7.26 Translation of event joins and temporal operators .......................... 109
<table>
<thead>
<tr>
<th>Listings</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.27 Creation of an interval event using global expiration time</td>
<td>110</td>
</tr>
<tr>
<td>7.28 Example translation for event aggregation over time windows</td>
<td>111</td>
</tr>
<tr>
<td>7.29 Example source code for lastsFor operator</td>
<td>111</td>
</tr>
<tr>
<td>7.30 Example target code for lastsFor operator</td>
<td>111</td>
</tr>
<tr>
<td>7.31 Example translation of player or avatar belongingness</td>
<td>112</td>
</tr>
<tr>
<td>7.32 Example translation of event consequences</td>
<td>113</td>
</tr>
<tr>
<td>7.33 Example translations assignment consequences</td>
<td>114</td>
</tr>
<tr>
<td>7.34 Example translations of deletion consequences</td>
<td>114</td>
</tr>
<tr>
<td>7.35 Compilation of TIME terminal</td>
<td>114</td>
</tr>
<tr>
<td>7.36 Translation of numeric expressions</td>
<td>115</td>
</tr>
<tr>
<td>8.1 Mapping of JSON-RPC call to corresponding Java interface method</td>
<td>128</td>
</tr>
<tr>
<td>8.2 Programmatic integration of notification widgets</td>
<td>132</td>
</tr>
<tr>
<td>8.3 Programmatic integration of mission widget</td>
<td>133</td>
</tr>
<tr>
<td>8.4 Initialization of mobile gamification widgets</td>
<td>133</td>
</tr>
<tr>
<td>8.5 Addition of gamification profile to host application</td>
<td>133</td>
</tr>
<tr>
<td>8.6 Addition of mission view to host application</td>
<td>134</td>
</tr>
<tr>
<td>8.7 GaML excerpt from the SAP TwoGo gamification concept</td>
<td>137</td>
</tr>
<tr>
<td>8.8 GaML excerpt from SAP Networking Lunch gamification concept</td>
<td>139</td>
</tr>
<tr>
<td>8.9 Example listing for simple GaML question</td>
<td>158</td>
</tr>
<tr>
<td>8.10 Example listing for complex GaML question</td>
<td>158</td>
</tr>
<tr>
<td>8.11 GaML excerpt from SAP Financial Fact Sheet application</td>
<td>166</td>
</tr>
<tr>
<td>8.12 Initialization example of mobile widgets</td>
<td>167</td>
</tr>
<tr>
<td>8.13 Initialization of mobile player profile widget</td>
<td>168</td>
</tr>
<tr>
<td>8.14 Initialization of mobile mission widget</td>
<td>168</td>
</tr>
<tr>
<td>8.15 Example instrumentation for sending events from host application</td>
<td>168</td>
</tr>
<tr>
<td>C.1 GaML example code for question block 3.1</td>
<td>225</td>
</tr>
<tr>
<td>C.2 GaML example code for question block 3.2</td>
<td>225</td>
</tr>
<tr>
<td>C.3 GaML example code for question block 3.3</td>
<td>226</td>
</tr>
<tr>
<td>C.4 GaML example code for question block 3.4</td>
<td>226</td>
</tr>
<tr>
<td>C.5 GaML example code for question block 4</td>
<td>227</td>
</tr>
<tr>
<td>C.6 GaML example code for question block 5</td>
<td>228</td>
</tr>
</tbody>
</table>
LIST OF ABBREVIATIONS

AIC      Akaike Information Criterion
API      Application Programming Interface
AST      Abstract Syntax Tree
AS       Achievement System
BEP      Business Entity Provider
BIC      Bayesian Information Criterion
BPEL     Business Process Execution Language
BPMN     Business Process Modeling Notation
BPM      Business Process Management
CBS      Component-based Software
CEGE     Core Elements of Gaming Experience
CEP      Complex Event Processing
CGDL     Card Game Description Language
CQL      Continuous Query Language
CRM      Customer Relationship Management
CV       Cross-Validation
DRL      Drools Rule Language
DSL      Domain-specific language
EBNF     Extended Backus-Naur-Form
ECA      Event-Condition-Action
ED-SOA   Event-driven SOA
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA</td>
<td>Event-driven Architecture</td>
<td></td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Beans</td>
<td></td>
</tr>
<tr>
<td>EPA</td>
<td>Event Processing Agent</td>
<td></td>
</tr>
<tr>
<td>EPL</td>
<td>Event Processing Language</td>
<td></td>
</tr>
<tr>
<td>EPL</td>
<td>Event Processing Language</td>
<td></td>
</tr>
<tr>
<td>EPN</td>
<td>Event Processing Network</td>
<td></td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning</td>
<td></td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
<td></td>
</tr>
<tr>
<td>ESP</td>
<td>Event Stream Processing</td>
<td></td>
</tr>
<tr>
<td>FCC</td>
<td>Fantasy-Challenge-Curiosity</td>
<td></td>
</tr>
<tr>
<td>GDL</td>
<td>Game Description Language</td>
<td></td>
</tr>
<tr>
<td>GGP</td>
<td>Generic Gamification Platform</td>
<td></td>
</tr>
<tr>
<td>GLiSMo</td>
<td>Serious Game Structure and Logic Modeling Language</td>
<td></td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
<td></td>
</tr>
<tr>
<td>IGS</td>
<td>Integrated Gamification Solution</td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td>Information System</td>
<td></td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
<td></td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
<td></td>
</tr>
<tr>
<td>JIT</td>
<td>Just-in-time</td>
<td></td>
</tr>
<tr>
<td>JMS</td>
<td>Java Messaging Service</td>
<td></td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming Directory Interface</td>
<td></td>
</tr>
<tr>
<td>JSON</td>
<td>Javascript Object Notation</td>
<td></td>
</tr>
<tr>
<td>LHS</td>
<td>Left-hand side</td>
<td></td>
</tr>
<tr>
<td>MDA</td>
<td>Mechanic-Dynamic-Aesthetics</td>
<td></td>
</tr>
<tr>
<td>MMRE</td>
<td>Mean Magnitude of Relative Error</td>
<td></td>
</tr>
<tr>
<td>MOM</td>
<td>Message-oriented Middleware</td>
<td></td>
</tr>
<tr>
<td>MRE</td>
<td>Magnitude of Relative Error</td>
<td></td>
</tr>
<tr>
<td>OASIS</td>
<td>Advancing Open Standards for the Information Society</td>
<td></td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraint Language</td>
<td></td>
</tr>
<tr>
<td>OLS</td>
<td>Ordinary Least Squares</td>
<td></td>
</tr>
<tr>
<td>PIM</td>
<td>Personal Information Manager</td>
<td></td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------</td>
<td></td>
</tr>
<tr>
<td>PIM</td>
<td>Platform Independent Model</td>
<td></td>
</tr>
<tr>
<td>PRED</td>
<td>Percentage Relative Error Deviation</td>
<td></td>
</tr>
<tr>
<td>PSM</td>
<td>Platform Specific Model</td>
<td></td>
</tr>
<tr>
<td>RHS</td>
<td>Right-hand side</td>
<td></td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
<td></td>
</tr>
<tr>
<td>RUP</td>
<td>Rational Unified Process</td>
<td></td>
</tr>
<tr>
<td>SCM</td>
<td>Supply Chain Management</td>
<td></td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
<td></td>
</tr>
<tr>
<td>SGDL</td>
<td>Strategy Game Description Language</td>
<td></td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
<td></td>
</tr>
<tr>
<td>SOA</td>
<td>Service-oriented Architecture</td>
<td></td>
</tr>
<tr>
<td>SOP</td>
<td>Same-Origin-Policy</td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
<td></td>
</tr>
<tr>
<td>SSO</td>
<td>Single Sign-On</td>
<td></td>
</tr>
<tr>
<td>SUS</td>
<td>System Usability Scale</td>
<td></td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
<td></td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
<td></td>
</tr>
<tr>
<td>VGDL</td>
<td>Video Game Description Language</td>
<td></td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
<td></td>
</tr>
<tr>
<td>WAR</td>
<td>Web Archive</td>
<td></td>
</tr>
<tr>
<td>WME</td>
<td>Working Memory Element</td>
<td></td>
</tr>
<tr>
<td>XSRF</td>
<td>Cross-Site-Request-Forgery</td>
<td></td>
</tr>
<tr>
<td>XSS</td>
<td>Cross-Site-Scripting</td>
<td></td>
</tr>
</tbody>
</table>
ABSTRACT

Gamification is a novel method to improve engagement, motivation, or participation in non-game contexts using game mechanics. To a large extent, gamification is a psychological- and design-oriented discipline, i.e., a lot of effort has to be spent already in the design phase of a gamification project. Subsequently, the design is implemented in information systems such as portals or enterprise resource planning applications. These systems act as mediators to transport a gameful design to its users.

However, the efforts for the subsequent development and integration process are often underestimated. In fact, most conceptual gamification designs are never implemented due to the high development costs that arise from building the gamification solution from scratch, imprecise design or technical requirements, and communication conflicts between different stakeholders in the project.

This thesis addresses these problems by systematically defining the phases and stakeholders of the overall gamification process. Furthermore, the thesis rigorously defines the conceptual requirements of gamification based on a broad literature review. The identified conceptual requirements are mapped to a domain-specific language, called the Gamification Modeling Language. Moreover, this thesis analyzes 29 existing gamification solutions that aim to decrease the implementation efforts of gamification. However, using the different language elements, it is shown that none of the existing solutions suffices all requirements.

Therefore, a generic and reusable platform as runtime environment for gamification is proposed which fulfills all presented functional and non-functional requirements. As another benefit, it is shown how the Gamification Modeling Language can be automatically compiled into code for the gamification runtime environment and, thus, further reduces development efforts.

Based on the developed artifacts and five real gamified applications from industry, it is shown that the efforts for the implementation of the gamification can be significantly reduced from several months or weeks to a few days. Since the technology is designed as a reusable service, future projects benefit continuously with regards to time and efforts.
ACKNOWLEDGEMENT

In the first place, I would like to thank my supervisors Professor Schill and Professor Zarnekow for supporting this thesis and its innovative topic and I would like to appreciate all of their extremely helpful input to get this work done in the desired quality and on time.

Deep gratitude goes to my supervisors and colleagues Michael Ameling, Bernhard Wolf, Christof Momm, Thomas Springer, and Ralf Ackermann. Thank you for all the fruitful discussions, white board sessions, paper reviews as well as all the organizational support to make this happen. An additional thank you to Bernhard Wolf for the complete review, correction, and improvement of this thesis document. This was an tremendous effort and provided fantastic improvements.

Further gratitude to the gamification platform development fellows: Axel Schröder (UI, still waiting for my Selenium tests), Kay Jugel (Backend), Martin Knechtel (Mobile), and Benjamin Heilbrunn (Advocatus Diaboli) who all contributed to this project in their spare time. Thank you, this was an awesome and highly intrinsically motivating experience to work with you.

For the various application scenarios I would like to thank the following SAP colleagues and fellows: Peter Graf, David Sommer, Jens Lehman, Dirk Lehmann, Uwe Steigmann, and Andreas Esch for supporting all phases of the SAP TwoGo Gamification project from the idea to the productization. Robert Wetzold and Melissa Visintin for providing a sophisticated gamification concept and designing beautiful badges for the SAP Networking Lunch application. Jens Körner and Alexander Manger for helping with the gamification of the SAP Financial Fact Sheet application. Finally, a big thank you to the SAP Value Prototyping department, in particular, Rumiana Petrova, Nils Heeren, Michael Dell, and Pascal Bräutigam for their endless efforts in positioning the gamification platform within SAP and implementing many custom applications and prototypes on top of it (e.g., CHIO Aachen, Soccer application).

Furthermore, I would like to thank my students Daniel Schlachter, Svenja Brunstein, Alexander Manger, Aneeque Hassan, and Johannes Schad for their excellent contributions and investigations that go beyond the content of this thesis.

I would like to thank my parents Heike and Gunther Herzig for their endless faith and patience. Finally, the biggest thank you goes to my girl Katjana Schneider who tremendously supported me over the last years. No doubt, this thesis would not exist without her endless patience as well as her remarkable hours of listening, discussing, evaluating but most importantly motivating me to continue the journey.


1 INTRODUCTION

1.1 MOTIVATION

Since early 2011, the interest of practitioners and researchers in gamification as a novel method for improving user behavior over information systems grew almost exponentially measured by the number of published papers and articles\(^1\). The majority of the work is concerned with the design, behavioral, or psychological aspects of the domain, i.e., researchers and practitioners investigate novel ideas, concepts, prototypes, models, or theories around human behavior and motivation. According to multiple studies, gamification has a strong potential to shape user behavior and influence human psychology positively (e.g., [91, 94, 95, 189]). Therefore, this thesis does not question if and how gamification works, but assumes that the concept, if applied correctly, works.

Although the gamification of enterprise information systems is a promising approach to support users’ motivation, enjoyment, and efficiency on the job, introducing gamification into existing or new systems is an expensive task with regards to development efforts. Simultaneously, the benefits for gamification are difficult to guarantee and to measure which makes the entire project a risky undertaking. The risk for enterprise systems is even higher due to their tight coupling with business processes, organizational structure, or the business model they support [118].

Nonetheless, there is little to no research investigating the specific technological requirements, methods, and tools that are necessary for successfully implementing gamification in information systems. In practice, most conceptual designs are never or only partially implemented due to the high development costs (e.g., [154]). In order to reduce initial development efforts as well as maintenance costs in the long run, so-called gamification platforms emerged on the market providing some features of gamification as a consumable service.

However, this thesis shows that existing approaches are not sufficient with regards to the functional and non-functional requirements. In particular, the requirements, features, methods, and architectural approaches of existing gamification solutions often remain unclear or unstructured. Different solutions exist for various specific purposes. This makes it arguably difficult for IT experts who are in charge of planning, executing, and operating the gamification project to select the technology or approach that fits best to the conceptual requirements from gamification designers.

\(^1\)Based on Google Scholar’s annual publication counts.
Therefore, this thesis rigorously investigates the technical requirements for gamification and realizes them in a generic service platform that enables the introduction of gamification in arbitrary information systems with high flexibility, integrability, and reusability.

1.2 RESEARCH OBJECTIVE AND RESEARCH QUESTIONS

So far, there is only little research regarding the implementation of gamification in information systems. In particular, a generic gamification platform is missing which supports all requirements of the domain and allows to maintain the gamification design in a flexible way.

Therefore, the overall research objective of this thesis is to investigate methods and tools that systematically support the development and implementation process for gamification and that reduce the associated efforts and costs.

More precisely, this thesis aims at answering the following research questions that are derived out of the general research objective:

1. Which gamification concepts, structures, and relationships exist and have to be provided by a generic gamification platform?

2. How can these concepts be represented in a formal way which is, nonetheless, understandable for gamification and domain experts?

3. Which services, components, and structures are necessary to constitute a generic platform for gamification which supports all identified concepts of research questions one and two?

4. How can the identified services and components be seamlessly integrated with arbitrary enterprise information systems?

1.3 RESEARCH DESIGN

For this thesis, constructivism has been chosen as epistemological assumption, because the resulting artifact does not exist naturally in any known setting but has to be constructed prior to evaluation. Hereby, the artifact in question is derived deductively out of existing theories and prior work based on a coherent and logic argumentation. Besides the derivation itself, the artifact is evaluated in controlled experiments with regards to resource utilization. Moreover, the artifact is tested in five independent and different real-world applications with regards to feasibility, completeness, applicability, and universality.

1.4 THESIS STRUCTURE

This thesis is structured as presented in Figure 1.1. First, the foundations for gamification and enterprise information systems are described in Chapter 2. To structure and classify the research questions and associated contributions, the gamification process is introduced in Chapter 3 as a modified software development process. Based on a comprehensive literature review, Chapter 4 structures and describes the conceptual requirements of gamification which are then formalized by the Gamification Modeling Language (GaML), a novel domain-specific language, in Chapter 5.
Figure 1.1: Overall structure of this thesis
Based on the syntactic elements and features of the language, existing technologies are classified and assessed in Chapter 6. This assessment demonstrates that none of the existing technologies suffices all technical requirements determined by GaML. Hence, a novel runtime environment capable of fulfilling the requirements is proposed in Chapter 7. Furthermore, Chapter 7 demonstrates the automatic compilation of GaML into the runtime environment’s model, i.e., describes how valid instances of GaML can be automatically compiled into executable code for the runtime environment.

Finally, all proposed concepts and artifacts are evaluated with regards to feasibility, applicability, and non-functional requirements using five gamified real-world applications or enterprise information systems (Chapter 8). Furthermore, one selected application is used to validate and demonstrate the benefits of the overall approach. The thesis closes with a summary on the research contributions and an outlook to future work.

1.5 PUBLICATIONS

This thesis consolidates a number of research results which have been published in journals and national or international venues. Hence, the content of this thesis does not develop novel results per se, but rather gives a comprehensive story using the isolated artifacts and outlines further details of the published work. In particular this concerns the following publications:

1.5.1 CONFERENCE AND JOURNAL PAPERS


1.5.2 PATENTS


2 FOUNDATIONS

In this chapter, relevant foundations and premises for the entire thesis are described. First, definitions and theoretical aspects of gamification are introduced and discussed. Second, technical topics, namely regarding service-oriented and event-driven architectures, are presented.

2.1 GAMIFICATION

In the following section, the topic of gamification is introduced presenting definitions, motivation, and psychological concepts.

2.1.1 DEFINITION

In the research literature two competing definitions exist for the term gamification. [65, 66] define gamification as

Definition 1. “the use of game design elements in non-game contexts.”

Their work is rooted in prior established theories. For example, the definition is based on Caillois [46, p. 27] who distinguishes between paidia and ludus as two kinds of different activities. While paidia (playing) refers to free-form, expressive, and improvisational behaviors, ludus (gaming) characterizes rule-based playing under determined goals in accordance with Caillois’ definition of a game:

Definition 2. “[...] an activity that is voluntary and enjoyable, separate from the real world, uncertain, unproductive in that the activity does not produce any goods of external value, and governed by rules” [46, p. 4].

This distinction is consistent with classical game research where ludus is characterized through explicit rule systems and discrete goals and outcomes (e.g., [107, 171]). Furthermore, McGonigal defines four basic traits that can be found in any game: clear goals that give the player a sense of purpose, rules that define the limitations how to achieve the goal, a feedback system giving the player the promise that the goal in question is definitely reachable and, ultimately, a voluntary participation, i.e., the user accepts the goals, rules, and feedback of the system voluntarily. Everything else, such as interactivity, narrative context, graphics, or rewards are enhancements or reinforcements of these defining features [136].
Consequently, [136] has introduced the term *gamefulness* in contrast to *playfulness* (e.g., [1]) for gamification. Past human-computer interaction research has mainly focused on the playfulness of software systems [65, 145]. However, research with regards to gamefulness received less attention in the past, although the idea of game element isolation and adoption is not completely new (e.g., [48, 49, 133]).

It is important to mention that the separation into paidia and ludus is a theoretical consideration. In practice, however, games and gamified systems may also coincidence with playful behaviors and attitudes (e.g., [22, 88]) and vice versa (e.g., [171]).

Besides this distinction, the definition implies that gamification is not about constructing games as a *whole* but merely applying certain parts (game design elements) thereof which may support motivation and participation in non-game contexts. Therefore, behind the definition a two-dimensional schema is assumed as presented in Figure 2.1.

Despite their rigorous definition, [65] do not report a complete list of game design elements which are specific to gamification. Instead, the authors provide a general taxonomy for structuring game design elements on various levels. In this taxonomy, exemplary design elements, tools, and methods of games are distributed across five abstraction levels including *game interface design patterns*, *game design patterns and mechanics*, *principles and heuristics*, *models*, and *methods of game design* (Table 2.1).

This hierarchy defines very generic and abstract methods (e.g., playtesting or play-centric design) on the highest level $L_5$ and very concrete elements (e.g., graphical game design elements) on the lowest level $L_1$ where the lower ones are created using the methods and tools of the higher levels.

A part of this hierarchy is consistent with classical game research. For example, [13] considers a *semiotic layer*, i.e., “the part of the game that informs the player about the game world and the game state through visual, auditory, textual, and sometimes haptic feedback.” This refers to the $L_1$ level of Table 2.1. Furthermore, [13] distinguishes a *mechanical layer* which “is the engine that drives the game action, [...] and changes the game state” and refers to $L_2$ of the presented hierarchy.

Moreover, the game mechanics layer ($L_2$) can be specified in more detail. According to [132], game mechanics “refer to the gameplay which emerges from [game] rules” and, therefore, may consist of one to many different rules. These rules may form a rule set procedure that determines the overall algorithm of the game [103] or be

![Figure 2.1: Classification of gamification (based on [65])](image_url)
2.1 Gamification

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game Interface Design Patterns (L1)</td>
<td>Common successful interaction design components and design solutions for a known problem in a context, including prototypical implementations</td>
<td>Badges, Leaderboards, Levels (e.g., [60, 202])</td>
</tr>
<tr>
<td>Game design patterns and mechanics (L2)</td>
<td>Commonly reoccurring parts of the design of a game that concern gameplay</td>
<td>Time Constraints, Limited Resources, Turns (e.g., [31, 68])</td>
</tr>
<tr>
<td>Game design principles and heuristics (L3)</td>
<td>Evaluate guidelines to approach a design problem or analyze a given design solution</td>
<td>Enduring play, clear goals, variety of game styles (e.g., [173])</td>
</tr>
<tr>
<td>Game models (L4)</td>
<td>Conceptual models of the components of games or game experience</td>
<td>Mechanic-Dynamic-Aesthetics (MDA) [123], Fantasy-Challenge-Curiosity (FCC) [133], Game design atoms [35], Core Elements of Gaming Experience (CEGE) [47]</td>
</tr>
<tr>
<td>Game design methods (L5)</td>
<td>Game design-specific practices and processes</td>
<td>Playtesting, Playcentric design, Value conscious game design (e.g., [83, 171])</td>
</tr>
</tbody>
</table>

Table 2.1: Levels of game design elements [65, p. 12]

considered as the “procedures of actions” [177], i.e., the procedures that are triggered based on the users’ input to the game.

[98] criticize the definition of [65] since it implies that every system qualifies as gamification as long as it features at least one game design element. Hence, they propose to include the overall goal of gamification into the definition, i.e., the support of the user’s overall value creation:

Definition 3. Gamification refers to a process of enhancing a service with affordances for gameful experiences in order to support user’s overall value creation [98].

Additionally, [98] consider gamification as a continuous process of enhancing a service or system with the respective game design elements.

In this thesis, both definitions are considered as valid. First, the isolation and application of game design elements in non-gaming context is considered as gamification. However, the current research literature does not provide a set or taxonomy of such game design elements. In fact, neither have researchers agreed on a common taxonomy of game elements yet [31, 173] nor has a particular subset of elements been translated into the domain of gamification in a central place. Thus, this thesis develops and evaluates its own taxonomy in conjunction with the levels provided by Deterding et al. In Chapter 4 the current gamification literature is surveyed, consolidated, and structured accordingly.

Second, this thesis considers gamification as a continuous process of enhancing and improving the companies’ business processes and information systems with regards
to participation, motivation, and engagement. This process itself is elaborated in more detail in Chapter 3. The consideration of this overall process is helpful to identify and understand practical problems which are later on addressed by this thesis.

Moreover, the presented definitions are considering non-game contexts in general. However, in this thesis the non-game context is narrowed down to enterprise or enterprise information systems such as supply chain management, customer relationship management, portals, or enterprise resource planning which act as mediator to transport a gameful design to its users in an inexpensive and scalable way. Hence, any other kind of gamified system which is not realized as an executable computer program or systems that are intended to apply gamification to other domains are excluded by this assumption.

### 2.1.2 Emergence and Progression Aspects of Games

Another important aspect of games is the consideration of emergence and progression which have to be reflected for gamification as well.

On the one hand, emergence is defined as the combination of rules that may lead to variation in games and is the primordial game structure according to [106]. In other words, games need rules that lead to many different combinations and states as in classical board or card games (e.g., chess). These potentially large numbers of combinations may then lead to the design and derivation of strategies. Consequently, the replayability of emergent games is high. Furthermore, emergent games are characterized through an end state (e.g., a winning state), i.e., the game is finite.

On the other hand, progression is characterized through a set of rules that lead to a comparably low number of states and which enforce the players to go through a predefined sequence of actions along a predefined path. According to [106], game mechanisms of progression have become popular with the rise of video games. For example, typical progression game genres such as jump-and-run or role playing games can be found almost in the form of video games only. Consequently, the replayability of such games is low and an end state is not necessarily required. Additionally, extending the game by new elements (e.g., missions, goals, or levels) is simpler compared to emergent games [106].

In the context of this thesis it has to be noted that gamification is primarily concerned with the design of progression games. It is argued that the non-game context in which the game design elements are applied is already an emergent system (e.g., the workplace), i.e., the gamification designer does not have to design primarily for emergence. This does not mean that the gamification design cannot be designed for or lead to emergent effects. However, it is not the primary design goal. Hence, it is notable that especially game mechanics of progression can be found in the existing gamification literature as presented in Chapter 4.

### 2.1.3 Biology & Psychology of Games

Neuroscience researchers found that playing video games releases high amounts of dopamine in the human brain [114]. Dopamine is associated with increased learning, reinforcement of the current behavior, and attention. Furthermore, [26] found that dopamine influences the incentive salience in general reward situations, i.e., the recipients want the experience more often. However, there is no mediation with hedonic impact, i.e., the recipients do not necessarily like the actual experience more.

Among others, dopamine is released in reward situations. Those rewards can be either of extrinsic or intrinsic nature. The former is represented by, for example, money,
status, goods, promotions, or punishment whereas the latter by, for instance, positive emotions, individual strengths, or social connections. While *extrinsic* motivators lead to *hedonic* behavior, *intrinsic* rewards lead to *autotelic* behavior under which all self-motivating and self-rewarding activities are subsumed [61]. [136] argues that those autotelic activities engage people completely and are the most “pleasurable, satisfying, and meaningful emotional states that we can experience.”

In fact, video games and its designers rely heavily on intrinsically motivating factors since extrinsic ones are virtually not available. For this experience, 58% of the American population spent $14.8 billion on video games in 2013 only excluding classical games [74]. From this percentage, more than five million people in the United States are playing games for 40 hours per week [69]. [163] found that 46.6% of employees play games during their working hours. As another example, gamers have collectively spent 5.93 million years of playing World of Warcraft [136].

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Antecedent of Work Engagement?</th>
<th>Antecedent of Game Engagement?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hope</td>
<td>Personal resource</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Optimism</td>
<td>Personal resource</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Resilience</td>
<td>Personal resource</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Mastery</td>
<td>Personal resource</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Self-efficacy</td>
<td>Personal resource</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Autonomy/Control</td>
<td>Job resource</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Social Support</td>
<td>Job resource</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Feedback</td>
<td>Job resource</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Skill variety</td>
<td>Job resource</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Environmental Climate</td>
<td>Job resource</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Positive Emotions</td>
<td>(e.g., enjoyment, awe, prosocial emotions, curiosity, enthusiasm)</td>
<td>YES</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of intrinsic factors leading to work and game engagement (based on [94])

The advantage of the provided intrinsically motivating factors is that they are, on the one hand, inherently associated with the perceived experience and, on the other hand, are created from oneself, i.e., they scale better compared to extrinsic rewards which are limited with regards to frequency and size.
In the field of psychology, researchers from the fields of positive psychology are studying constructs that lead to happiness. The adoption of concepts from positive psychology into the field of organizational psychology led, in addition, to the proposal of happiness conceptualizations (e.g., engagement) in workplace environments.

Established theories and models are, for example, the self-determination theory [169], the job demand-resource model [17, 18, 64], psychological capital [131], positive organizational behavior [130], or flow [62, 96]. As an example, the job demand-resource model describes intrinsic factors leading to engagement on the job. These factors are divided into personal resources (e.g., hope, mastery, or self-efficacy) and job resources (e.g., social support, skill variety, environmental climate).

Subsequent research compared such factors leading to engagement on the job with factors leading to engagement in games [94]. It has been found that the majority of factors are supported equally in both domains (Table 2.2).

This comparison supports the working hypothesis of gamification in the enterprise, i.e., that it is possible to foster higher levels of engagement and motivation through the isolation and application of game mechanics in this particular domain. Based on these factors, various scientific studies demonstrated the quantitative or qualitative benefits of gamification (e.g., [91, 94, 95, 189]).

This concludes the consideration of various aspects of gamification as a foundation for this thesis. It has been shown how gamification differs from classical and video games. Furthermore, models, theories, and hypotheses of gamification have been outlined. As a general premise of this thesis, it has been defined that the gamification of the workplace is considered herein. Chapter 3 introduces the corresponding process and the key roles for introducing gamification in enterprise information systems. Based on this process, this thesis specifically considers the technological aspects that are necessary to investigate to introduce gamification in large-scale enterprise scenario. Those foundations are presented in the following section.

### 2.2 ENTERPRISE ARCHITECTURES

In most cases, gamification is implemented as an additional aspect into a set of information systems of the enterprise, e.g., Enterprise Resource Planning (ERP), Supply Chain Management (SCM), Customer Relationship Management (CRM) and other types of systems, which are pervasively used in the workplace today. Those systems provide an adequate means to transport and mediate a holistic gameful experience to its end users.

However, introducing gamification into existing or new systems is an expensive task with regards to development efforts. Simultaneously, the benefits for gamification are difficult to guarantee and to measure which makes the entire project a risky undertaking. The risk for enterprise IS is even higher due to their tight coupling with business processes, organizational structure, or the business model they support [118].

Therefore, this section examines general architecture paradigms and their inherent characteristics and requirements which build the technical foundations for the rest of this thesis.

#### 2.2.1 OVERVIEW

The authors in [118] argue that enterprise software completely differs from other software systems, such as system-software, desktop applications, or video games, because it is strongly connected with the internal organization of the enterprise as well
2.2 Enterprise Architectures

Requirement Description

Simplicity The enterprise architecture must be simple in order to allow efficient communication between key personnel.

Flexibility and Maintainability Due to changes in, e.g., economies, laws, or reorganization, the enterprise system must be highly flexible to be adopted in new contexts or changeable to new requirements.

Reusability The architecture should provide basic building blocks that can be reused in a variety of contexts.

Decoupling of functionality and technology The architecture has to abstract from technologies and implementation details in order to make the decisions in the enterprise independent from it. This goal aims to easily integrate innovative technologies by seamlessly migrating from the old ones.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicity</td>
<td>The enterprise architecture must be simple in order to allow efficient communication between key personnel.</td>
</tr>
<tr>
<td>Flexibility and Maintainability</td>
<td>Due to changes in, e.g., economies, laws, or reorganization, the enterprise system must be highly flexible to be adopted in new contexts or changeable to new requirements.</td>
</tr>
<tr>
<td>Reusability</td>
<td>The architecture should provide basic building blocks that can be reused in a variety of contexts.</td>
</tr>
<tr>
<td>Decoupling of functionality and technology</td>
<td>The architecture has to abstract from technologies and implementation details in order to make the decisions in the enterprise independent from it. This goal aims to easily integrate innovative technologies by seamlessly migrating from the old ones.</td>
</tr>
</tbody>
</table>

Table 2.3: General requirements for enterprise systems (based on [118])

as its business processes and model. Therefore, enterprise software has to deal with specific requirements. The overall goal of enterprise architectures and software is to provide the agility, flexibility, and efficiency that is necessary to fit the constant change of an organization. In fact, in practical embodiments, the number of change requests for a particular enterprise solution is negatively correlated to the agility of the entire system over time [118]. Reasons are either of technical or, more importantly, organizational nature. Based on their observations, the same authors provide general requirements for such systems shown in Table 2.3.

In prior research two main design methodologies have emerged that try to fulfill these requirements on an abstract level: Service-oriented Architecture (SOA) and Event-driven Architecture (EDA). In the subsequent text, both methods are described. Moreover, related concepts which emerged out of these two principles, such as Business Process Management (BPM) or Complex Event Processing (CEP) are presented for the sake of completeness.

2.2.2 SERVICE ORIENTED ARCHITECTURE

Service-oriented Architecture (SOA) and its related design principles are one of the most discussed topics in the context of enterprise software [124, 179]. However, there is no commonly accepted definition of the term itself [24, 75, 108]. The Advancing Open Standards for the Information Society (OASIS) group attempts to define standards in the context of SOA. Therein, SOA is defined as “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains” [143].

In this context, the capabilities are provided via services which are defined as “a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description” [143].

Another definition is given by [149, 150] who define services as “autonomous, platform-independent entities that can be described, published, discovered, and assem-
bled and are technologically neutral, loosely-coupled and support location transparency encapsulating business functionality.

Although no common definition exists, different researchers share the same view on service-orientation. This view includes, for example, autonomous and interoperable services that offer reusable business functionality via standardized interfaces [124]. Moreover, these services should be loosely-coupled and provide the possibility to be orchestrated within business processes [191]. Additionally, services may have different levels of abstraction and appear on various layers within a holistic application. For example, services may reflect entire processes, the presentation layer, business logic, or data management functionality.

Figure 2.2 presents an exemplary service-oriented architecture of an airline. First, on the lowest layer basic services such as a flight, customer, or booking services exist which are used to maintain and retrieve data of the underlying business entities.

Second, more complex services exist on an interim layer which hides the communication complexity that might be necessary between multiple basic services. Usually, the services are façades, gateways, adapters, or functionality enhancing services [118, pp. 108].

Third, services may be composed within business processes. In this case, the services are executed in a particular order whereas the communication complexity between services, users, and the surrounding context is hidden by respective BPM tools. In this example, the booking process is instantiated after a customer has selected a flight from the basic flight service. Afterwards, the customer’s data might be retrieved or created using the customer service. Subsequently, the booking itself is issued and an invoice is sent upon completion of the process instance.

Fourth, the website of the airline is the central entry point for customers to use and interact with the underlying enterprise services.

1Please note that the term logic in this thesis refers to a particular functionality, procedure, or algorithm and does not refer to any kind of predicate logic (e.g., first-order, second-order, many-sorted, or infinitary logic).
The intention of this approach is to increase modularity and reusability of components which is a major software design goal in computer science. However, the granularity of such services is usually higher than in traditional software components (e.g., libraries, frameworks) since they may cover larger blocks of functionality on the one hand or even rely on subsequent services on the other hand. It is also important to note that the different services need not to belong to the same owner but can be provided by different teams, departments, or other companies [104].

Ultimately, SOA accepts the wide heterogeneity of distributed systems and parts thereof that can be usually found in Information Technology (IT) landscapes of companies [104, pp. 14]. This heterogeneity arises from the variety of utilized platforms, programming languages, or technologies. With the introduction of services and their interface descriptions, service users can fully abstract from the underlying implementation details.

Based on this argumentation, Table 2.4 summarizes the most important traits of services that can be found in the literature.

<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardized</td>
<td>Services within the same service inventory are in compliance with the same contract design standards</td>
</tr>
<tr>
<td>Contract</td>
<td></td>
</tr>
<tr>
<td>Loose Coupling</td>
<td>Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment</td>
</tr>
<tr>
<td>Abstraction</td>
<td>Service contracts only contain essential information and information about services is limited to what is published in service contracts.</td>
</tr>
<tr>
<td>Reusability</td>
<td>Service contain and express agnostic logic and can be positioned as reusable enterprise resource.</td>
</tr>
<tr>
<td>Autonomy</td>
<td>Services exercise a high level of control over their underlying runtime execution environment.</td>
</tr>
<tr>
<td>Statelessness</td>
<td>Services minimize resource consumption by deferring the management of state information when necessary.</td>
</tr>
<tr>
<td>Composability</td>
<td>Services are effective composition participants, regardless of the size and complexity of the composition.</td>
</tr>
</tbody>
</table>

Table 2.4: Characteristics of services (based on [76, p. 49])

In the airline example above, the need of service composability has been already visualized. In this sense, various services are wired together to achieve a more complex goal. Therefore, closely related to SOA is BPM as the overarching discipline to manage the company’s business processes, for instance, travel reimbursements or sales processes. Such business processes can be defined, for example, as a “collection of activities performed by human users or software applications that together constitute the different steps to be completed to achieve a particular business objective” [7].

BPM tools or engines allow to take a particular business process definition (e.g., via Business Process Modeling Notation (BPMN) or Business Process Execution Language (BPEL)) at design-time and create running instances of this processes by creating a
formal model internally. Those formal models are, for example, based on Petri Nets or Pi-Calculus [195]. The running process receives event from its users or the service it invokes. Moreover, the BPM monitors the processes at runtime, handles exceptions, or helps power users to administrate and maintain the running instances.

With regards to enterprise gamification, such business processes are intended to be gamified in order to achieve the motivational improvements described in Section 2.1. In terms of SOA, the service infrastructure, therefore, has to be extended by a gamification service which provides the desired characteristics. This thesis aims at a technical gamification solution that can be provided in the sense of the service notion described above. Thus, the considered preliminaries provide the foundations for examining the non-functional requirements of the ideal gamification service in Section 4.3.

2.2.3 EVENT DRIVEN ARCHITECTURE

Event-driven Architecture (EDA) has emerged as a comparably new term for a specific class of architectures and can be considered as complementary to SOA. As some of its aspects are used within this thesis (e.g., Chapter 7), necessary foundations are considered in the following.

2.2.3.1 DEFINITION, CHARACTERISTICS, AND BUILDING BLOCKS

Event-driven Architecture (EDA) is defined as “an architectural style in which some of the components are event driven and communicate by means of events” [129, p. 16]. According to [126], an “event is an object that is a record of an activity in a system defined through the following three aspects:

1. Form, data of the event represented by a tuple of arbitrary data objects
2. Significance, the event form should contain data that signifies the event.
3. Relativity, an activity is related to other activities by time, causality, and aggregation” [126, p. 88].

EDA is especially useful when one or more of the following requirements are necessary: First, maintaining complex logic such as pattern matching, aggregation of homogeneous and heterogeneous events across applications, or correlation of current and historical data. Second, processing of high volume data and events which arrive continuously with a low sampling period. Third, computation of results should be available in (soft) real-time. Fourth, scalability is required along increasing amounts of data or transactions. Finally, agility is necessary to react quickly on changing business demands and requirements [39, p. 39].

As the definition of EDA implies, components are event-driven by default, i.e., they start processing upon reception of events and send the results of their calculation to one or more subsequent receivers in the form of events. A puristic EDA distinguishes the following types of components [39].

Event sources: An event source is a special component which creates events from relevant information. Those are packaged into messages and sent to a mediator component. It is important to mention that the event source makes no assumption about where the event is consumed or if it is consumed at all [39]. Hereby, all entities such
as entire applications, business processes, internal or external stakeholders of an enterprise might be considered as event sources or generators [124].

**Event sinks** Event sinks are arbitrary complex software components which are able to receive events [129] and are invoked upon event arrival. By definition, sinks contain the business logic to process the events.

**Event mediators** Mediator components are dispatching events from event sources to all interested sinks. Here, not only one-to-one but one-to-many communication between event sources and sinks is possible. A mediator may also queue events over a predefined time window when event sinks are overloaded or not present. Furthermore, components may interact in a publish-subscribe pattern, i.e., multiple sources act as publishers and sinks act as subscribers of events. The subscription can be, for example, based on channel, content, or topic. In practical embodiments, the term Message-oriented Middleware (MOM) is used in literature to describe the technology which realizes the mediator’s functional and inherent non-functional requirements (e.g., [200]). Term ambiguity is introduced by further terms such as Enterprise Service Bus (ESB) which offers concepts beyond message passing (e.g., [39, p. 163]). Nonetheless, ESBs include the basic message mediation and passing functionalities of MOMs as well. Figure 2.3 presents the above mentioned components and their dependencies schematically.

![Figure 2.3: Components of an EDA (based on [39, p. 52])](image)

**Event processors** In addition to these three basic building blocks of EDAs, event processors can be considered as a fourth component. The upper perspective presumes that each event contains all relevant information for the components to proceed. Thus, so far one event at a time has been considered only. However, [39] argue that independent considerations of single events are not sufficient in most application domains. Identifying dependencies, relationships, and correlations (all subsumed as patterns below) between events brings up interesting information. Under this premise, different methods such as reactive rule systems, CEP, or Event Stream Processing (ESP) aim at providing fast detection for interesting patterns in continuous event streams or unstructured event clouds. Hence, the following two subsections describe some fundamentals for these approaches.

### 2.2.3.2 COMPLEX EVENT AND STREAM PROCESSING

According to [126], “CEP enables to identify causal, temporal, and spatial relationships between events. Those relationships specify patterns that are analyzed in real-time using event-pattern matching.” CEP can be also considered as a process of event selection, aggregation, hierarching, and abstracting of events to generate higher level
events [152, p. 59]. [39] argue that “CEP is an integrated part and key element of any technically sophisticated event-driven architecture”.

In reactive and event-driven systems such as CEP, the traditional data processing as used in database technologies is inverted. In contrast to databases where transient queries are issued against persistent data, with CEP, persistent queries are issued against steady, continuous, and dynamic data or event streams. This implies that queries are of reactive nature. Thus, zero latency can be assumed between event arrival and event processing which supports the basic requirement of real-time behavior in event-driven systems.

CEP is an evolutionary approach since its development roots in prior work such as production rule systems [176], active databases [153], and event/action/transition logic systems [172]. In the last years, CEP has evolved as the overarching discipline summarizing the aforementioned approaches and technologies [152, p. 58]. Sometimes the term Event Processing Agent (EPA) is used instead of CEP which describes the entire technology stack that is necessary to receive, process, and send events. Multiple EPAs may form a network in order to distribute the execution (e.g., for scalability, fault tolerance, complexity reduction, or maintainability) of events which is, consequently, referred to as Event Processing Network (EPN). In this context, an EPN is inherently distributed with strong cohesion and loose coupling of the single components, while a monolithic EPA has weak cohesion and is strongly coupled with regards to functionality [39, pp. 71-72].

Moreover, CEP has to be distinguished from ESP. ESP refers to technologies which analyzes ordered and continuous data streams with regards to predefined event patterns [39, 129]. In this context, ESP emphasizes the efficient extraction, aggregation, and modification of events with regards to high throughput and low latency [198]. To deal with high volume data, operators in stream processing systems act on finite sliding windows to convert blocking operators such as \textit{sum}, \textit{count}, or \textit{max} into non-blocking ones [52] and to output results continuously, i.e., for each newly arrived event an output event is generated. Furthermore, these operators primarily process events on their payload and attributes, e.g., to average the temperature attribute of events received from a machine.

On the other hand, CEP mainly focuses on complex event pattern detection in arbitrary event clouds [152, p. 59], i.e., there is not explicit notion of sliding windows [52]. Instead, the utilized operators, their associated semantics, and the events’ timestamps determine how long events are of interest. This is referred to as context or consumption mode [52, 53]. Additionally, events are typically correlated based on the tuple level rather than the attribute level, i.e., based on their occurrence timestamps and pure existence, for example, using logical operators, temporal, or spatial operators as shown below. Since there is no general definition how CEP and ESP formally differ from each other, hereinafter, ESP methods and tools are considered broadly as a real subset of CEP.

Figure 2.4 presents schematically how complex events are constructed from simpler events in a CEP engine. First, so-called simple or atomic events arrive at the CEP engine. Those events are characterized through instantaneous occurrence, i.e., the event’s duration equals zero. On this layer, selection operators are used to filter events based on their attributes.

Second, some events from the receiving layer might be aggregated into more complex events or they are enriched with further data (e.g., the event gets a duration or
the event gets additional attributes as payload\(^2\). Again, events might be filtered based on the resulting payload. Furthermore, temporal operators might be applied when the events are equipped with a duration. Overall, there are 13 theoretical operators to correlate events temporally as defined in [6]. Figure 2.5 shows the semantics of four different operators, namely during (a), after (b), meets (c), or finishes (d) [10]. All diagrams comprise two events called A and B having a start timestamp (i.e., the event’s occurrence) and an end timestamp (i.e., the event’s occurrence plus its duration). Furthermore, the operator refers always to event A as the left operand and event B as the right operand, e.g., event A happens during event B. Note that operators expressing the logical negation are available as well. In this example, the semantically opposite operators cover include, before, metBy, and finishedBy respectively.

![Figure 2.5: Exemplary Temporal Operators [6]](image)

Third, the same selection, enrichment, aggregation process might be continued until all events have been processed accordingly.

Languages for CEP foster a declarative writing of rules and are subsumed under Event Processing Languages (EPLs). In most cases, well-established query languages such as Structured Query Language (SQL) have been extended by language constructs to support, e.g., sliding time windows, stream elements, or temporal operators [117, p. 13].

For example, there are SQL-based languages [3, 12, 117] as used in systems such as Streambase [184] or Esper [77] as well as the Continuous Query Language (CQL)

\(^2\)Since events are considered immutable, in fact, enriching refers to the enrichment of an event’s copy herein.
in general. Furthermore, rule-based EPLs exist such as Reaction RuleML [167], Tibco [190], or the Drools Rule Language [8, 70]. Finally, there are agent-oriented EPLs, e.g., RulePoint [100] or EventZero [78], and scripting languages, e.g., Netcool Impact [99] or Apama [159].

2.2.4 PRODUCTION RULES

Today’s event processing systems often combine event stream or complex event processing systems with logical rules to enable further application classes [52]. Production rules in the form of if-then rules are used in some practical implementations on top of the event processing layer. Such rule systems have a longer history in so-called expert systems or, more recently, business rule management systems (BRMS) (e.g., [166, 196]). The main idea of this approach is to externalize the business logic from the rest of the application in order to increase the agility of the infrastructure to meet fast changing business requirements. In accordance with [166], structuring the logic of the enterprise in business rules helps to avoid ad-hoc rules which lead to confusion, contradiction, operational inefficiency, miscommunication, rule inconsistency, and inaccessible rules. Moreover, business rules are entitled to speed up operational efficiency, since logic can be adopted or changed faster.

Such logical rules or productions consists of one condition or Left-hand side (LHS) which comprises arbitrary set of logical expressions or patterns and an action or Right-hand side (RHS) which is executed when the condition becomes true. Furthermore, there is a set of Working Memory Elements (WMEs) or facts residing in the main memory and which comprise several attribute-value pairs. The rules are executed on top of the current WMEs. To solve the matching problem between multiple rules on the one hand and multiple facts on the other hand, an efficient algorithm, called RETE, has been proposed [81].

In this algorithm, all rules and their patterns are decomposed and translated into a directed graph representation as shown exemplarily in Figure 2.6. In the classical approach, $\alpha$, $\beta$, and terminal nodes are distinguished [81]. New or updated WMEs from the fact base are propagated as tokens through this graph.

In the first step, tokens pass the root node. Second, $\alpha$-nodes process patterns related to intra-element structures. For example, the graph splits up into multiple $\alpha$-nodes when multiple values are checked on the same attributes (in the example, the element’s type). Different attribute checks are furthermore wired together sequentially. These $\alpha$-nodes are stateless as they only have to consider the currently received token and, if the criterion matches, propagate the token to the next node. Consequently, these nodes have exactly one input and output node and do not modify the processed token.

Third, $\beta$-nodes process inter-element patterns, e.g., join operations between different tokens. Thus, these nodes have two, a left and right, input. Moreover, they are equipped with a left and right internal token store. In the given example, selected tokens of type $A$ are received on the left input and tokens of type $B$ on the right input. Within the node, these tokens are joined. For example, if a token of type $A$ is received on the left input, the token’s ID is matched against all tokens IDs from the right internal memory where previously received tokens of type $B$ have been stored. For all successful equi-joins, new tokens are generated which are propagated to the subsequent node, i.e., one input token may lead to many output tokens on a $\beta$ node. Finally, the received token is stored within the node’s left internal memory. As a consequence of this process, $\beta$ nodes are always stateful and their runtime complexity depends on the size of the internal token stores and, hence, on the history of processed propagations.
Eventually, tokens pass the criteria of all nodes and reach a terminal node. Hereby, each terminal node represents the corresponding business rule. If the token reaches the terminal node, the production is added to the conflict set, a structure where all rules which have been activated in the same evaluation cycle are collected and then executed to a corresponding resolution strategy [21, 89].

As some rules may utilize the same patterns in their LHS, the graph creation algorithm can detect this situation and reuse nodes across different rules, a technique called node sharing and which speed-ups the pattern matching process further [67].

In general, RETE’s theoretical runtime complexity for one firing lies between logarithmic execution time, i.e., $O(1 + \log_2 P)$ in the best case and polynomial execution time $O(W^{2C-1} + P)$ in the worst case with $W$ being the number of facts in the fact base, $C$ being the number of patterns, and $P$ being the number of productions. The worst case applies for the propagation time of one token under the assumption that no discrimination network exists (i.e., no $\alpha$ nodes) and that $C$ equal inter-element patterns (e.g., join
operations) have to be performed consecutively on $\beta$ nodes whose internal memories are all filled up with $W^C$ tokens [80, p. 98]. Hence, both, the best and worst, cases are irrelevant for practical scenarios. Despite some investigation (e.g., [5, 67]), the average case has not been sufficiently determined yet as it depends on the history and statistical distribution of processed tokens and, therefore, the state of the nodes’ internal memories at execution time. Nonetheless, researchers and practitioners agree that the worst case is not only very unlikely but also irrelevant for practical considerations as it is orders of magnitude larger than the average case [20].

Hence, rule engines with the RETE approach provide an adequate means for the design of fast and reactive systems that allow for high degrees of flexibility and agility. As already argued by [52], rule engines can be used in conjunction with CEP or ESP systems. For example, the Drools technology combines all aforementioned approaches in one single solution.

2.2.5 CONTEXTUAL EVENT PROCESSING

Although formulated as one of the fundamental requirements by [183], the processing of stored and persistent data in conjunction with volatile event-driven systems has received less attention in research [144]. In fact, traditional expert systems and rules engines process facts from a volatile working memory. Event storages or sliding windows in CEP or ESP systems are volatile as well, i.e., in a failure situation all data might be lost. Furthermore, those systems are not able to take additional contextual knowledge into account, e.g., from ontologies, databases, or other non-streaming data systems (e.g., [3, 12, 137, 182]). Hence, for some application classes, various researchers raised the question if the existing systems are “[…] expressive enough to capture complex (business) events in all their aspects. How likely is that critical decisions are taken merely on event patterns such as: event a is followed by event b in last 10 seconds?” [11, p. 123]. Therefore, additional and more persistent knowledge might be necessary upon event arrival to answer more complex questions. Thus, both approaches have been integrated by numerous authors and are commonly known under Stream Reasoning [194] or Knowledge-based CEP [8, 10].

A generic solution has been proposed in [144]. The authors utilize a Business Entity Provider (BEP) that defines business entities encapsulating complex state derivation logic and data management functionality. To interact with this data, the BEP offers query and update interfaces to an Event Processing Agent (EPA). After rules have been activated by external event sources in the EPA, the rules’ consequences may issue one or more updates to the entities managed by the BEP. Subsequently, the BEP calculates a new entity state based on the update operation, e.g., sum the value from the current operation with values from prior updates. Moreover, a state change in any entity can be propagated back to the EPA continuously or on demand. In the former case, the respective entity is continuously monitored for state changes and the change is inserted into the event agent as additional event. In the latter case, the entity provider is queried by the EPA once a new event arrives from the event sources. This behavior is visualized in Figure 2.7.

This approach provides two main advantages compared to previous ones. First, durable and persistent entity state can be managed across the application’s life-cycle. This state can be used while processing events to answer more complex questions. Second, the definition of complex state-calculation logic can be encapsulated in business entities, i.e., they do not have to be defined as rules in the EPA anymore. This
does not only reduce the complexity of the rule base in the EPA, but allows for the distribution and definition of “standardized” logic independently of rules [144].

In this thesis, this approach is translated and extended to the domain of gamification in Chapter 7. Furthermore, the authors of the approach did not provide quantitative evidence that it works in practice and fulfills the general requirements of event-driven applications. In fact, it is shown in Chapter 8 that this general scheme possesses major drawbacks with regards to performance. Hence, further extensions and resulting trade-offs have to be discussed.

Other related concepts in this domain comprise, for example, [185] which allows the encapsulation of state in procedural definitions locally available to the stream operators in an ESP system. Moreover, [116] proposed well-encapsulated state management. In their approach, the state is maintained at the context in which it is processed. State deltas are managed and distributed by a state transformer. However, both approaches dictate a certain style of programming paradigm, i.e., procedural or functional programming. [186] proposed semantic CEP for knowledge rich event processing, i.e., the usage of ontologies together with rules in order to allow rules such as “buy Stocks of Companies, who have production facilities in Europe and produce products from Iron and […] and their price/volume increased stable in the past 5 minutes”. Their main assumption is that the ontology does not fit into the working memory and that the ontology is rarely updated by external authorities. Based on this assumption, they proposed several strategies to query the ontology. However, besides a polling strategy, other strategies are tailored to the specifics of semantic query languages. Due their specific assumptions and target platforms, these approaches are of lesser interest in this thesis.

This concludes the consideration of the foundations on event-driven systems. This chapter closes with a systematic comparison of the architectural styles and their implications for the rest of this thesis.
2.2.6 COMPARISON OF APPROACHES

More recently, all of the aforementioned concepts are not perceived as competitive approaches but complementary ones [127, 128].

Both paradigms have in common that they are not architectures themselves, but design paradigms leading to a concrete architecture. As such, both principles aim at improving the general requirements for enterprise architectures as outlined in Section 2.2.1. However, they are achieved by different means and trade-offs. Hence, Table 2.5 compares both approaches regarding a selection of aspects.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>SOA</th>
<th>EDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus of Design</td>
<td>Interfaces of Services</td>
<td>Events and Messages</td>
</tr>
<tr>
<td>Modularity</td>
<td>Services &amp; Clients</td>
<td>Event Sources, Sinks &amp; Middleware</td>
</tr>
<tr>
<td>Distributability</td>
<td>Technology-inherent</td>
<td>Technology-inherent</td>
</tr>
<tr>
<td>Communication</td>
<td>synchronous, Request/Reply</td>
<td>asynchronous, publish/subscribe, multiple recipients</td>
</tr>
<tr>
<td>Area of Transaction</td>
<td>Client and Services form one logical unit of work</td>
<td>event source and sinks form independent logic unit of works</td>
</tr>
<tr>
<td>Relationship between components</td>
<td>Client has to know address and interface of service</td>
<td>Components do not know each other. Event sources make no assumption about sinks.</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Interface definition, version, and SLAs</td>
<td>Event syntax</td>
</tr>
<tr>
<td>Coupling</td>
<td>Loosely coupled</td>
<td>Extreme loosely coupled</td>
</tr>
</tbody>
</table>

Table 2.5: Comparison of characteristics in service-oriented and event-driven architectures (based on [39, p. 37])

In contrast to the service-oriented approach, an EDA fosters the asynchronous communication between enterprise components rather than traditional synchronous request and response patterns. Hence, components are not explicitly called by an interested party but react on arbitrary events. Of course, asynchronous Remote Procedure Call (RPC) communication between services is also possible but not an often deployed feature [39].

Moreover, the loose coupling is increased within EDAs since interface descriptions have not to be known in advance. In contrast to SOA, participants are completely decoupled from each other which results in increased flexibility and agility of the entire architecture (e.g., [41]).

On the other hand, EDAs suffer from the clear definition of how components, interfaces, or data contracts are structured. Hence, SOA principles may help in EDA contexts to successfully structure all system instances as services. Therefore, researchers proposed to combine both approaches into so-called Event-driven SOA (ED-SOA) [32, 127, 128, 147, 180].
For the rest of this thesis, this perspective is shared, i.e., components are designed in the sense of services with predefined and structured interfaces and data contracts. Additionally, these components will communicate by the means of the events and the introduced event-driven methods are used to detect patterns and achieve soft real-time behavior across services, thus, enabling the overall service landscape to react at least in soft real-time.

This comparison closes the considerations of foundations. In the following chapter, the general gamification process is introduced by describing its constituting workflows and roles involved into the associated tasks. Based on this process the contributions of this thesis are defined in more detail.
3 GAMIFICATION PROCESS

In this chapter the general gamification process is introduced. The purpose of this process is to visualize how gamification is introduced stepwise into arbitrary existing information systems starting at the business modeling phase and ending at the monitoring and improvement phases. Furthermore, the contributions of this thesis are positioned in the context of the presented process.

3.1 GENERAL PROCESS

In this thesis, the introduction of gamification into Information Systems (ISs) is understood as a modified software development process. Hence, the gamification process is rooted in a well-known method, namely the Rational Unified Process (RUP) as proposed by [119]. Due to its generic nature, the RUP has been translated into more specific workflows. For example, Figure 3.1 shows a combined version of the plain RUP with a Component-based Software (CBS)-approach as proposed by [54]. Within the depicted process each box represents one workflow comprising multiple tasks and roles. The thin arrows represent the flow of artifacts between workflows, i.e., artifacts that result out of a workflow are input to one or more subsequent workflows. The gray arrows, finally, indicate that there is a bilateral change of activity, e.g., between users involved in the various tasks and workflows.

In this process, the following phases are considered. First, the business modeling phase in which all participants of the development process should get a common understanding of the business processes that the information system should support. Second, the requirements phase in which the general business requirements for the solution are collected, refined, and evaluated. They constitute the formal set of requirements and restrictions for the design phase. Furthermore, the relevant target group and the general vision and goals for the project are elicited. Third, the design phase in which the software components are specified and the general architecture of the components is created. Fourth, the provisioning phase in which reusable components are identified and provisioned, i.e., these components might be purchased or licensed from external parties or procured company-internally from other departments. Furthermore, documentation and Application Programming Interfaces (APIs) are collated. Fifth, the implementation phase in which existing components are assembled and missing components are implemented as determined by the design artifacts produced in the design phase. Sixth, the testing phase, where the whole system or parts thereof are tested.
with regards to use cases, requirements, design specification and architecture. Finally, the deployment phase takes places where the system is put into productive mode and used by end users.

### 3.2 GAMIFICATION PROCESS

The general RUP can be adapted to the domain of gamification. Therefore, this section explains the actions that take place in each workflow. Within these workflows different roles have specific responsibilities. Therefore, the following roles are defined for this thesis:

- **The end user** is a role which is shared by a group of people who are experiencing the gamification in the end. They participate in a set of identified business processes and should engage with these processes better and more often after gamification has been introduced. Depending on the context, the end users are either employees (in a business-to-employee context) or customers (in a business-to-customer context) who are participating in internal or external processes of the company respectively.

- **Gamification experts** are persons with high expertise in designing compelling and engaging game or gamification designs. Furthermore, those persons have very good knowledge of psychological models and methods as well as general game design methodologies and tools. Preferably, persons who own this role have already designed a couple of successful gamification applications before.
Domain experts are persons with profound knowledge of the target business processes and their target end users. Domain experts should have high understanding on the positive and negative aspects that end users are experiencing within the target processes. Furthermore, the ideal person for this role is responsible for the processes and, thus, interested in its improvement.

Business experts are persons who are responsible for the overall project and have to manage the project’s budget, deadlines, and stakeholders. Overall, these persons are responsible for the successful execution of the project and to achieve its initial objectives.

IT experts preferably have high expertise in designing, provisioning, assembling, implementing, and deploying large scale IT-systems in the company. These persons exactly know the existing IT landscape of the company and are responsible for the seamless integration of new components and tools into the existing infrastructure. Moreover, these persons advise domain, business, and gamification experts on the decision for the right software components and tools to successfully deploy the gamification solution on top of the existing business processes.

Based on these roles, Figure 3.2 presents an adapted version of Figure 3.1, specific to gamification projects. The individual phases are described in the following text.

BUSINESS MODELING

The business modeling workflow has the same task as in the RUP. The domain experts explain the set of business processes which are intended to be gamified to all other involved stakeholders except end users. The goal of this phase is that all participants get a common understanding of the business processes including their merits and drawbacks. Moreover, the general objectives of the project have to be identified. This includes the type of end users who should be engaged (employees, customers, external stakeholders) and other important environmental variables. Finally, the project’s vision and the goals are communicated to all project members.

REQUIREMENTS

In the requirements phase, the respective use cases are analyzed based on the project’s goals. Based on the business modeling phase, the identified target group has to be analyzed with regards to motivation, engagement, and participation within the target processes. This analysis can be conducted qualitatively or quantitatively, for example, using interviews or questionnaires. The analysis has to cover at least the question, what the target user group motivates already or generally to participate in the processes and what are reasons to not participate. As a result of this analysis a list of pros and cons for the participation in the target business processes as well as motivating factors should be compiled.

Besides the analysis of the current situation, business, domain, and gamification experts should explicitly agree on the target situation and the metrics that measure the project’s success, for example, what should the outcome of the gamification be (e.g., 100% increased user retention).

Key personnel involved in this workflow includes at least: gamification experts, domain experts, business experts, and end users.
3 Gamification Process

<table>
<thead>
<tr>
<th>Step #</th>
<th>Step Name</th>
<th>Responsible Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Creation/Revision of gamification concept</td>
<td>Gamification experts</td>
</tr>
<tr>
<td>#2</td>
<td>Presentation of gamification concept</td>
<td>Gamification experts</td>
</tr>
<tr>
<td>#3</td>
<td>Proposal of modifications to design concept</td>
<td>Domain experts or Business representatives</td>
</tr>
<tr>
<td>#4</td>
<td>Prepare gamification concept for playtesting</td>
<td>Gamification Experts, IT experts</td>
</tr>
<tr>
<td>#5</td>
<td>Playtesting the current concept with end users</td>
<td>Gamification experts</td>
</tr>
</tbody>
</table>

Table 3.1: Sequence of tasks in the design and specification workflow

DESIGN

The design phase is primarily concerned with the specification of a meaningful game or gamification design. From the design perspective, it is the most complicated one since all stakeholders have to agree on a specific and precise design alternative in the end. This phase, furthermore, is supposed to have multiple iterations as known from classical game or application design (e.g., [171]).

In each iteration the gamification concept is refined and revised by the gamification experts based on the outcome of previous iterations. In the first step, an initial gamification draft is proposed based on the agreements and findings of the business modeling and requirements phases.

Second, the proposed or revised concept is presented to the domain and business experts using presentation slides or spreadsheets. Third, if these stakeholders propose modifications to the concept, the gamification experts start over from the first step and revise the gamification concept accordingly. If, however, these stakeholders approve the concept, the design is ready for the fourth step which comprises the preparation of playtesting sessions [171]. For this preparation the gamification experts may either develop the gamification alone using a low-fidelity prototype or consult IT experts to implement a proof-of-concept as software solution (high-fidelity) already.

Fifth, the prototype of the gamification is play-tested with the end user group which shares its opinion to the design, identifies problems, and proposes improvements based on the experience with the prototype. These suggestions and improvements lead back to the first step and may result in a revised version of the gamification concept.

This workflow can be continuously repeated until all stakeholders have agreed that the gamification concept might be an appropriate solution to the problems identified in the requirements workflow. Output of this workflow is the precise definition of the intended gamification concept. The specification workflow is presented in Table 3.1.

PROVISIONING

The provisioning phase is adopted from [54]. In this step, the IT experts take the gamification design and the requirement specification in order to analyze the market for appropriate gamification solutions (Chapter 6). This decision is influenced by technical constraints of the existing IT infrastructure, the gamification concept itself, and the process models that emerged as deliverables out of the previous steps. Hence, IT
3.2 Gamification Process

Figure 3.2: Adapted gamification development process

experts are responsible for the execution of this workflow. In some cases, they may consult domain, gamification, or business experts to clarify the gamification concept or to discuss additional technical constraints with regards to the gamification concept.

Output of this workflow is the provisioning of all systems that are necessary to integrate including their documentations, APIs, and tools. Nonetheless, the output of this workflow might also be that no gamification solution can be reused and that a custom implementation has to be provided. In this case, the output of this workflow comprises further the right resources which allow for the successful implementation of the gamification concept. This may include additional resources out- or inside the company.

IMPLEMENTATION

After the provisioning phase, the IT experts are responsible to assemble, implement, and integrate all components for the final gamified ISs. If the provisioning phase resulted in the reuse of integrated, generic, or achievement solutions (see Chapter 6), this phase concerns the integration of host applications executing the target business process with the respective gamification solution. Furthermore, the gamification concept has to be created and configured within the selected gamification solution.

If, however, the IT experts decided against the reuse of existing solutions, the IT department has to implement the gamification on its own without involvement of external services. Therefore, implementation may take place within the existing applications or as independent part providing appropriate APIs for integration. Furthermore, additional services might be assembled and integrated such as identity, user management, or e-
mail services. Additionally, custom user interfaces have to be implemented to present the gamification to the end user in an easy to use way.

In this workflow the IT experts of the company are primarily involved. Outputs of this workflow are the gamified applications in a first running prototype.

**TEST**

The testing phase joins all the artifacts from the prior workflows and tests requirements and assumptions against the running prototype. This includes technical tests for functional correctness and non-functional quality attributes. Additionally, this workflow includes domain, gamification, business experts, and end users to test whether the application behaves as designed or not. Output of this workflow is the tested gamified application.

**DEPLOYMENT**

If all tests have been passed successfully, the IT experts are responsible for the final deployment in the IT landscapes and to open access to all end users. Furthermore, invitations or change logs might be sent to all stakeholders informing them of the successful deployment of the solution.

**MONITORING**

The RUP assumes an end-to-end process, i.e., the process stops after the deployment phase. However, in the context of gamification additional steps are necessary after deployment. This primarily concerns the monitoring of the gamification after deployment. The deployment phase continuously outputs operational user data signifying the usage of the target process. This data is assembled and aggregated in the monitoring phase in accordance with the engagement criteria and process models defined by the requirements phase. Based on the comparison of these three aspects, additional modifications and suggestions for improvement might be derived. Those improvements are communicated as delta to the design phase which starts the entire process again.

Figure 3.2 presents the adapted gamification development process.

**3.3 CONTRIBUTIONS**

Based on the adapted gamification development process, the contributions of this thesis are summarized.

First, a mechanism for the precise definition of gamification concepts in the design phase is missing. Currently, gamification concepts are discussed in natural language transported through natural text, spreadsheets, or presentations (e.g., [36, 189]) or on a very formal mathematical level which is not understandable to domain or gamification experts (e.g., [29]). Since gamification is an inter-disciplinary method, it is argued that this makes the interchange and discussion of game mechanics between experts of different domains complicated, especially in business environments. For example, handing an informal concept over to an IT expert for the provisioning and implementation phases is an error-prone task since the concept may miss crucial information for the formal realization. Furthermore, it hardly allows the exchange on game mechanics on a rigorous, formal level between researchers, for example, in discussions, related work, or meta-reviews. Therefore, a novel, domain-specific and declarative language is
proposed in Chapter 5 which formalizes the conceptual requirements on the one hand and addresses the design phase problem on the other hand.

Second, the implementation of the gamification concept within an IS is an expensive task with regards to development efforts. At the same time, the benefits for gamification are difficult to guarantee and to measure which makes the entire project a risky undertaking. Gamification platforms emerged on the market to reduce this risk and effort (e.g., [16, 43]). However, existing systems for gamification have not been analyzed with regards to functional and non-functional requirements for enterprise gamification. This makes it difficult within the provisioning phase of each gamification project to select the appropriate gamification solution. Therefore, Chapter 6 analyzes and classifies existing solutions with regards to requirements and present inherent trade-offs of the existing solution classes.

Third, the analysis of related work (Chapter 6) yields that existing technologies focus on very simple game mechanisms and are, therefore, not suitable for implementing more sophisticated concepts as required by gamification in general. Due to these limitations, platforms dictate the conception phase of the gamification concept to some degree since they impose their particular language on the concept. This results in a tight coupling of concept and requirements with the underlying technology which has to be avoided in enterprise settings [118]. Hence, a runtime technology supporting all required features is missing. Therefore, Chapter 7 of this thesis describes a novel runtime environment concept which is capable of executing the conceptual requirements based on the proposed formal gamification language.

This closes the consideration of the gamification process. In the next chapter, the functional and non-functional requirements for gamification are analyzed as prerequisites for the contributions described above.
4 CONCEPTUAL REQUIREMENTS

In this chapter, the conceptual requirements (CR) for gamification are analyzed. To identify proper requirements, a qualitative literature review has been conducted. These requirements are described informally in this chapter and are formalized in Chapter 5.

This chapter is structured as follows. First, the approach for the literature review is described. Second, based on the literature all identified game design elements on L1 and L2 of Deterding’s structure are described. Third, the non-functional requirements for supporting gamification technologies are discussed. The purpose of this analysis is to build the foundations for related work discussion and to design the contributions of this thesis later on.

4.1 APPROACH

The literature review was conducted in the second quarter of 2012. Scientific paper databases were used to identify papers which contain the term gamification within their title or text body. The analysis was conducted using the databases of ACM Digital Library, Springer, IEEE Xplore Digital Library, Google Scholar, ScienceDirect, and EBSCOHost. This analysis yielded a total of potential 918 publications.

From these papers, only those were selected which went through a scientific peer-review, have been edited and published by well-known authorities (i.e., self-published work such as technical reports or student theses were discarded), and are available in English language. Moreover, only those papers remained which incorporate a discussion on at least one game design element transported to a non-gaming context in the presented approach, i.e., papers which make only a reference to gamification (e.g., in the outlook) were withdrawn. Finally, very short academic papers (e.g., position papers with less than three pages) were also omitted. The selection process narrowed the literature down to 37 relevant publications.

The remaining ones were sorted into four classes. First, papers that are aiming at a definition of the term gamification based on prior work (e.g., [65, 66, 88, 90]).

Second, quantitative papers which observe the effects of gamification towards user behavior or psychological outcomes (e.g., [29, 95, 178, 189]).

Third, papers which introduce gamification into non-gaming systems but without studying the outcome with regards to any engagement criteria (e.g., [36, 87, 135]).

Fourth, papers or books that consolidate the state-of-the-art, especially subsuming and elaborating on the game mechanics that are relevant for gamification ([44, 44, 65, 35]).
90, 109, 136, 141, 162, 197, 202]). It is important to note that the union of the identified books covered all the elements used in the papers of the second and third category. In fact, most of the papers are merely using advancing points, badges, and leaderboards as their defining mechanisms. The existing books, however, look at the topic more broadly and identify mechanisms which have not been deployed in research studies at this point in time. However, the books suffer from providing rigorous definitions for the individual mechanisms, clear taxonomic classification, or providing term disambiguation.

Therefore, the existing body of game research was used to define and disambiguate the different terms used in literature (e.g., [30, 31, 68, 106, 107, 173, 201]). Although the literature review was carefully examined, a vast amount of publications appeared afterwards. Therefore, [91] published a more recent literature review at the beginning of 2014. It is important to note that the analysis of their paper, however, does not list any elements beyond the ones described herein. Hence, it is concluded that the described functional requirements of this thesis are at least consistent with the current consensus in research.

### 4.2 General Gamification Requirements

As already argued, game and gamification researchers have not agreed on a common taxonomy of game elements yet [31, 173]. Therefore, in this section a literature review is presented that identifies common elements. To structure these elements, the coarse-grained taxonomy of [65] as presented in Table 2.1 (p. 9) is used. In this taxonomy five levels for gamification design elements are introduced: Game interface patterns (L1), Game Design Patterns (L2), Game Design Principles (L3), Game Models (L4), and Game Design Methods (L5).

L1 comprises visual and tangible elements (e.g., points or notifications) which can be directly and explicitly experienced by an end user [13]. The L2 level comprises the game mechanics or rules of the game [68, 103, 177]. Those can be experienced directly, however, in an implicit way only as their structure has to be derived by the user based on the explicit L1 elements.

All other levels (L3-L5) comprise design methods to create compelling games or gamification designs. Therefore, they are considered as less important herein since they do not directly influence the creation of the intended generic gamification service. The purpose of the analysis below is to derive common elements from literature and to structure them on the L1 and L2 levels.

#### 4.2.1 Game Interface Patterns

For the first level of Deterding’s taxonomy, the identified visual concepts of gamification are described below and summarized in Table 4.1. For each element its name, possible synonyms, subtypes of the element, and references to its occurrence in literature are provided.

Furthermore, this level is divided into two sub-levels. First, so-called atomic concepts are considered, i.e., concepts which exist for themselves and cannot be dissected further into other atomic gamification concepts. Second, aggregated concepts are analyzed, i.e., concepts which subsume at least two or more atomic gamification concepts.
4.2 General Gamification Requirements

4.2.1.1 ATOMIC CONCEPTS

Points (CR1): According to [202], points are “required for all gamified systems”. As such, they are operationalizing fundamental aspects of the game, for example, to express progress in various dimensions. Further subtypes of Points can be found such as advancing, redeemable, reputation, karma, and skill points [202]. Advancing points are considered as points that always increase, i.e., they cannot decrease at any point in time. Redeemable points might be considered as a virtual currency which can be traded for real or virtual goods. Karma points are points which are shared between players only, i.e., they are a special form of redeemable points which are traded without the involvement of goods. Skill points are a special type of redeemable points as they can be used to improve virtual skills that players have. Reputation points differ from all prior categories since they have a predefined range in which points can be assigned, e.g., in the range between one and five points. Finally, this thesis defines auxiliary points, i.e., specific metrics which are not explicitly presented to the users as points but serve as basis for tracking arbitrary things internally such as level progress or missions.

Achievements (CR2) While Points are an interval-scaled measure, Achievements are their nominal (without order) or ordinal (with order) counterparts. According to [30], achievements describe “goals whose fulfillment is stored outside the scope of individual game sessions.” Achievements are feedback for special or rare situations. The terms badges, trophies, medals can be treated as synonyms for achievements depending on the context or graphical representation [44, 90, 141, 202]. According to [90], an achievement is at least signified through name, visual badge, and description of the operational rules for getting the achievement. Subtypes for achievements are: visible achievements, i.e., the condition is visible to the user, hidden achievements, i.e., the condition for the achievement is not known to the users in advance, or riddle achievements, i.e., the achievement’s goal is only described in a fuzzy way.

Goods (CR3) Goods are objects that players can get for completing the various goals within the game. Here, virtual items and real goods are distinguished. While virtual items defined as “diegetic objects in game worlds that can be carried or interacted with” [30] are only subject to the game, e.g., the item might be used to give others or oneself some advantage in the gamification progress, real goods do not have any impact in the virtual space but might be used in the real world. Goods might be given to the user either directly as consequence of their actions or indirectly over redeemable points.

Skills (CR4) According to [173], skills are an essential part of any game. Skills can be distinguished into virtual and real skills where [173] primarily addresses real skills (e.g., social, physical, or mental skills) that are necessary to play the game in general. On the other hand, virtual skills (abilities) are “actions through which characters or units can affect game worlds” [31]. In this thesis, virtual skills are considered as instances from real skills, i.e., the virtual skill in the gamification solution may reflect real skills of the employees in the workplace. As stated by [202], these skills are usually correlated with skill points expressing various levels of the skill in question.

Roles (CR5) Roles (or functional roles) are used where “responsibility for different types of game actions can be divided between participants” or players [30]. Typically
4 Conceptual Requirements

A role determines a predefined set of skills. Furthermore, in some cases a role might have a graphical representation, e.g., visualizing a specific character or class thereof.

Missions (CR6) Missions define what has to be achieved within the game which “adds purpose, focus, and measurable outcome” [109]. Moreover, missions have to be concrete, achievable, and rewarding [173, pp. 148-149].

According to [68], mission can be in three states: unavailable, available, and completed. A mission is considered to be available, if the player or team is allowed to execute its conditions. Furthermore, a mission is considered to be completed when all its conditions are fulfilled formally. Hence, single missions allow the definition of arbitrary complex gamification rules, i.e., conditions and consequences as described in Section 4.2.2. Besides mission completion, the individual conditions may lead to subconsequences such as points or badges.

In addition, the rules of missions are conveyed through a graphical representation to the user, so that users can interpret the underlying formal structure of the rule. This, for example, comprises the mission’s name, description, or completion consequence.

Besides their individual manifestation, progression games comprise several missions which are wired together in specific sequences or so-called mission graphs [68]. According to [68], topologies for levels and mission can be classified into three types: linear and directed paths (acyclic directed graphs), networks (cyclic graphs), and open worlds (undirected graphs) based on an analysis of, e.g., [2, 45, 168, 173].

Figure 4.1 presents the six building blocks for mission graphs.

![Figure 4.1: Constituting elements of mission graphs [68]](image)

There are three types of nodes, namely normal (a), entry (b), and end (terminal) missions (c). Furthermore, there are three types of edges, i.e., weak (d) and strong (e) requirements and inhibition (f). A new progression is started on at least one entry node, i.e., there is at least one mission available. Players can aim at fulfilling conditions of available missions. Once these conditions are fulfilled, the mission’s state is set to “completed”. Furthermore, a node is set to “available” when its preceding node is in available state and the edge is a weak requirement or the preceding node is in completed state and the edge is a strong requirement. Moreover, a node is set to “unavailable” when a preceding mission is in completed state and the edge is an inhibitor [68].

If a mission has multiple predecessors, the following semantics are defined: If there are strong prerequisites, then all of them have to be completed in order to set the mission’s state to available; else if there are weak prerequisites, then at least one of them has to be completed in order to set the mission’s state to available [68].

Figure 4.2 shows a valid instance of a mission graph with respect to the provided model. This example covers a structure of 14 tasks with one entry and terminal state.
Five missions have been already completed as indicated by the checkmarks. The upper subgraph is not accessible anymore due to the inhibition of the third task. The second last task in the middle requires the strong prerequisite on the lower subgraph to be completed.

---

Although mission graphs are a powerful means to express dependencies between missions, the proposal of [68] still has some inherent limitations. First, the semantics of a join, i.e., multiple missions stream into one single node, are limited to logical “and” for strong requirements and logical “or” for weak requirements. Thus, arbitrary and nested Boolean expressions are not possible. Second, the model proposes that a mission is always dependent on the fulfillment of another mission or nothing (in case of an entry node). However, additional preconditions (e.g., player state or randomness) cannot be taken into account.

Therefore, preconditions are considered more broadly in this thesis, i.e., a mission’s precondition should allow the reference of arbitrary contextual information. Finally, in this thesis it is distinguished between rule-driven mission and user-accepted mission. While the former refers to the automatic assignment of mission when they become available, in the latter case, the player has to explicitly agree on taking the mission. If the user does not agree, the mission’s state is set to “declined” or might be offered again at a later point in time.

**Feedback (CR7)** According to [136], feedback “gives the player the promise that the goal in question is definitely reachable” [136]. Furthermore, it helps to change behavior by improving factors such as self-efficacy, hope, and resilience (e.g., [14, 131, 162]). [109] further distinguishes informational and corrective feedback. First, informational feedback considers any kind of positive feedback when the user does the right actions under the given rules [109]. In the context of gamification, identified informational feedback mechanisms are: points, achievements, notifications, real goods, virtual goods, narration, and missions. Second, corrective feedback is given, if the user does not do the right actions [109]. Corrective feedback is, usually, given through notifications including a hint for improvement. Additionally, also the loss of prior achievements might be used as corrective feedback. In this case the user should derive the improvement implicitly, i.e., there is no explicit message that guides users into the right direction.

**Events (CR8)** Events are external or internal actions in the game space. Specifically, events can be distinguished into user actions, external, and internal events. While
user actions describe the set of activities the players can do in the solution (cf. [173]), external events occur without the influence of users, e.g., from an external system or the environment. According to [173], user actions can be further divided into operative and resultant actions. While operative actions describe the actions enforced by the rules of the gamification, resultant actions are of higher abstraction and emerge out of the operative actions. Moreover, events may exist that signify some interim state of the gamification, for example, to postpone a calculation to a later point in time. These events are called internal events herein.

<table>
<thead>
<tr>
<th>#</th>
<th>Game Design Element</th>
<th>Synonyms</th>
<th>Subtypes</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Point</td>
<td>Measure, Metric, Currency</td>
<td>Advancing, Redeemable, Karma, Reputation, Skill</td>
<td>[44, 197, 202]; [91]^1</td>
</tr>
<tr>
<td>CR2</td>
<td>Achievement</td>
<td>Badge, Trophy, Virtual Good</td>
<td>Expected, Unexpected, Partially (un-)expected</td>
<td>[44, 90, 141, 202]; [91]^1</td>
</tr>
<tr>
<td>CR3</td>
<td>Goods</td>
<td>Items, Stuff</td>
<td>Virtual, Real</td>
<td>[30, 44, 162, 202]</td>
</tr>
<tr>
<td>CR4</td>
<td>Skills</td>
<td>Abilities</td>
<td>Physical, Mental, Social</td>
<td>[31, 162, 202]</td>
</tr>
<tr>
<td>CR5</td>
<td>Roles</td>
<td>Functional Roles, Character Roles</td>
<td>-</td>
<td>[30, 162]</td>
</tr>
<tr>
<td>CR6</td>
<td>Mission</td>
<td>Goal, Challenge</td>
<td>Individual, Collective</td>
<td>[109, 173, 201], [91]^1</td>
</tr>
<tr>
<td>CR7</td>
<td>Feedback</td>
<td>None</td>
<td>Informational: Points, Notifications, Achievements, Narrative Context; Corrective: Notifications</td>
<td>[109, 136, 162]; [91]^1</td>
</tr>
<tr>
<td>CR8</td>
<td>Event</td>
<td>User Actions</td>
<td>Operative, Resultant, External, Internal</td>
<td>[173]</td>
</tr>
<tr>
<td>CR9</td>
<td>Narrative Context</td>
<td>Storytelling</td>
<td>-</td>
<td>[109, 162]; [91]^1</td>
</tr>
<tr>
<td>CR10</td>
<td>Notification</td>
<td>Alert</td>
<td>-</td>
<td>[65, 109, 173]; [91]^1</td>
</tr>
</tbody>
</table>

Table 4.1: Visual (basic) game design elements

^1Refers to a more recent survey which was conducted in 2014 by [91], i.e., after the main parts of this thesis were created. Their paper supports the surveyed concepts mentioned herein since multiple international papers use the concept in the context of gamification.
4.2 General Gamification Requirements

Narrative Context (CR9): Besides the various feedback mechanisms being already discussed above, gamification concepts may include a narrative context represented on L1. While, for example, notifications merely describe what has happened, narration might be used to tell what is going to happen in the future. As such narration can be used for many purposes such as reinforcing curiosity, attention, and engagement [9], giving hints [162], or to improve memory of the transported content [121]. Narrative messages can be generated either explicitly in gamification rules or indirectly on the assignment or completion of achievements, points, or missions.

Notifications (CR10): As already stated in the feedback paragraph, notifications as direct and immediate feedback mechanism are important to any gamification design [136]. Notifications can transport various content such as progress (e.g., upon the achievement of elements described above), positive messages, or hints for improvement to reinforce motivation [109]. Usually, notifications comprise a small text and icon that visualize the type of feedback directly.

4.2.1.2 AGGREGATED CONCEPTS

Further visual elements belonging to L1 can be identified from the literature. These elements are aggregations of the atomic elements according to some specific aggregation function. For example, an avatar is the aggregation of specific amounts of, e.g., points, badges, roles, or skills for a particular player. A leaderboard is the aggregation of avatars and their points etc. Some of these aggregations can be determined already at design-time, i.e., when the gamification concept is specified, whereas other aggregations are of abstract nature at design-time and emerge at the gamification’s runtime. All identified aggregations are described in the following text and systematized in Table 4.2.

Space (CR11) The space of the game or gamification “defines the various places that can exist in a game and how those are related to one another” [173, p. 130]. As such, the space is the most general element and comprises all elements in L1 as well as L2. Therefore, it is implicitly an element determined at design-time.

Game Levels (CR12) “A [game] level is a part of the game in which all player action takes place until a certain goal has been reached or an end condition has been fulfilled” [30]. Hereby, each game level represents a non-disjoint subset of the entire space and may include all or some L1 and L2 elements, i.e., some elements might be valid globally, while others are valid locally within the respective level. If defined, the game level is also determined at design-time.

Context (CR13) According to [173], the space contains objects having attributes and state. In this thesis, the state of the various objects is considered as contextual information of the object in question or other objects. Hereby, context comprises all atomic elements as well as information on the player. For example, the player’s current points, badges, levels, etc. are state information representing the context of the individual player. Furthermore, the state information of another player might be of interest, for example, in collaborative (e.g., trading, joint actions) or competitive scenarios (joint actions). Furthermore, the game’s entire state might be of interest to the gamification owner in order to observe the engagement of all players at the moment. Consequently,
the state as such is a runtime concept which cannot be explicitly modeled at design-
time. However, as shown later gamification rules query and update the context. Hence,
conditionals will refer to contextual information already at design-time.

**Avatar Levels (CR14)** The level (rank) of an avatar is determined through one or mul-
tiple point categories. With avatar levels the entire range of one point category is
divided into smaller buckets. Furthermore, each bucket is given a name which refers
to the level’s name. In some cases, the avatar level is also called level, player level
or character level (e.g., [30, 202]). Levels and ranks may directly influence trust and
reputation in that person [164] besides other metrics such as badges or points when
acquired under publicly known rules. Again, the specification of these levels is done at
design-time.

**Avatars (CR15)** An *avatar* is the virtual representation of any physical player in the
game [162] and fosters engagement and curiosity [125]. Therefore, the avatar assem-
bles all contextual information about the player. Furthermore, an avatar may have ex-
actly one role assigned or particular level based on the contextual information described
above. Moreover, there is an one-to-many relationship between players and avatars,
that is, one player may have multiple avatars at runtime.

Avatars can be also used as customization mechanisms [202], i.e., the player is al-
lowed to define an individual presence based on predefined or achievable virtual items.

**Marketplace (CR16)** A gamification design may comprise real or virtual currencies
and associated goods and, thus, may also enable marketplaces [162]. In this thesis,
currencies are considered as redeemable points as described above. Moreover, virtual
or real goods might be traded using marketplaces in the gamification concept. Hence,
the marketplace is an aggregation of goods, redeemable points, and players (avatars)
acting as traders. The marketplace itself is a runtime concept based on the specification
of the basic elements.

**Leaderboard (CR17)** A leaderboard is the aggregation of avatars’ or teams’ points
with respect to some specific point category in ascending or descending order. It is
used to compare players or teams against each other with regards to their points of
the same category [30, 202]. In this context, leaderboards enable the social interaction
of players in a competitive way [109]. While the content of the leaderboard is created
at runtime, the specification of the leaderboard (e.g., aggregation, time period, point
category) can be already defined at design-time.

**Communication System (CR18)** As another social interaction feature, communica-
tion systems can be adopted from games to gamification as a runtime concept (e.g.,
[162]). In this communication system, players or avatars can directly exchange mes-
sages and receive these messages as notifications from others. It can be used as
communication channel especially in collaborative and competitive scenarios. As this
concerns a runtime requirement, its realization is discussed in Chapter 8.

**Team (CR19)** Teams comprise a real subset of players under a specified team name
(e.g., [136, 162]). Teams are formed at runtime, i.e., they are not directly part of the
gamification concept. However, some goals might be accomplished by teams only and
the consequences for achieving the goal may affect all or some players of that team.
Other design alternatives include, for example, that goals are completed by an arbitrary player but that the consequences for fulfilling the goal apply to all members of the player’s team. This additional scoping aspect, therefore, has to be considered already at design-time.

<table>
<thead>
<tr>
<th>#</th>
<th>Game Design Element</th>
<th>Synonyms</th>
<th>Aggregates</th>
<th>Known at</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR11</td>
<td>Space</td>
<td>Gameplay</td>
<td>-</td>
<td>Design-time</td>
<td>[173]</td>
</tr>
<tr>
<td>CR12</td>
<td>Level</td>
<td>Game Level, Wave, Checkpoint</td>
<td>-</td>
<td>Design-time</td>
<td>[109, 201]</td>
</tr>
<tr>
<td>CR13</td>
<td>Context</td>
<td>Objects’ state</td>
<td>Points, Levels, Achievements, Goods, Skills</td>
<td>Runtime</td>
<td>[173]</td>
</tr>
<tr>
<td>CR14</td>
<td>Avatar Levels</td>
<td>Level, Player Level, Character Level, Rank</td>
<td>Points</td>
<td>Design-time</td>
<td>[44, 109, 202]; [91]</td>
</tr>
<tr>
<td>CR15</td>
<td>Avatars</td>
<td>Character</td>
<td>Context, Roles, Notifications, Avatar Levels, Goals</td>
<td>Runtime</td>
<td>[44, 109, 162, 202]</td>
</tr>
<tr>
<td>CR16</td>
<td>Market-place</td>
<td>Virtual Economy</td>
<td>Avatars, Virtual Goods, Virtual Currencies</td>
<td>Runtime</td>
<td>[162]</td>
</tr>
<tr>
<td>CR17</td>
<td>Leaderboard</td>
<td>Highscore, Scoreboard</td>
<td>Points, Avatars</td>
<td>Design- &amp; Runtime</td>
<td>[30, 44, 109, 202]; [91]</td>
</tr>
<tr>
<td>CR18</td>
<td>Communication System</td>
<td>None</td>
<td>Notifications</td>
<td>Runtime</td>
<td>[162]</td>
</tr>
<tr>
<td>CR19</td>
<td>Team</td>
<td>Cooperation</td>
<td>Avatars</td>
<td>Design- &amp; Runtime</td>
<td>[136, 162, 202]</td>
</tr>
</tbody>
</table>

Table 4.2: Aggregated visual game design elements

This list concludes the game design elements on L1 for this thesis. It is argued that this list might not be complete. For example, [30] outlines many additional patterns which can be assigned to L1. However, these patterns have been explicitly defined for games and have not been transported to the domain of gamification yet. Hence, further additions to the elements described above may emerge in future work. Therefore, the provided list of elements cannot be complete and has to be extended and reworked when new requirements become available.
4.2.2 GAMIFICATION RULES

Besides concepts on L1, concepts on L2 concern game design patterns determining how the gamification is “played.” According to [173]: “Rules are really the most fundamental [game] mechanic. They define the space, the objects, the actions, the consequences of actions, the constraints, and the goals.” While the concepts on L1 refer to the existence of concepts on a meta-level or describe their visual appearance to the users, rules determine the gamification logic and how instances of these concepts evolve over time.

Therefore, the following text describes possible characteristics of rules in the gamification domain. First, different types of rules are characterized whereas only foundational and operational rules are selected as important for this thesis. Second, the characteristics of these two rule types are explained in more detail.

4.2.2.1 TYPES OF RULES

In accordance with [171], rules limit player action, are explicit and unambiguous, are shared by all players, and are fixed, binding, and repeatable. Rules can be distinguished into operational rules, foundational rules, behavioral rules, and instructional rules [109, 171, 173]. Operational and foundational rules describe how the game has to be played and under which constraints the goals are achieved. Operational rules determine what the players have to do and what the outcomes of their behavior are. Foundational rules represent the underlying formal structure of operational rules, i.e., they reflect the implementation of rules and, thus, inform operational rules directly (cf. [173]). Behavioral rules are implicit rules, i.e., what is allowed or commonly accepted to do in the game (e.g., rules which emerge out of the community). Instructional rules are the rules why the game was created in the first place, i.e., these are rules that players should remember as a “lesson-learned” even if the game is already over.

In this thesis, only foundational and operational rules are considered as they constitute the formal rule set or are the rules which are presented to the player in descriptive manner. In the following, the basic elements of foundational rules are described. In general, foundational rules comprise at least one formal condition and consequence which is applied once the condition is fulfilled. Furthermore, rules should allow arbitrarily complex conditions and consequences.

4.2.2.2 RULE CONDITIONS

On the conditional side of rules, multiple elements from the L1 layer can be considered. This includes user or external events (e.g., user \( u \) does action \( a \) or mission completed during day or night), the current state or context of objects (e.g., user \( u \) has already 50 points or has skill \( s \) activated), information about avatars or their levels (e.g., user \( u \) has the novice level), or the current state between players (e.g., only top three players with regards to a particular point category). Second, besides the direct use of concepts from L1, further elements can be identified as described below.

Constraints (CR20) Constraints place the limitations under which the goals of some entity in question can be achieved (cf. [109, 173, 197]). The state of L1 concepts can be combined in arbitrarily complex patterns using Boolean (CR20a), numeric (CR20b), temporal (CR20c), or spatial (CR20d) constraints. Boolean constraints allow the recombination of L1 elements using logical operators (e.g., user \( u \) has to do action \( a \) and \( b \)).
4.3 Non-Functional Requirements

Numeric expressions allow the filtering for specific elements (e.g., user $u$ has to do a sales order action $a$ having a purchase amount greater than $30$).

Temporal operators allow for the filtering of events with regards to their temporal aspects. These temporal aspects can be evaluated over a set of heterogeneous events (e.g., user $u$ has to do action $a$ five seconds before action $b$), over a particular time window of homogeneous events (e.g., user $u$ has to do action $a$ five times in one week), or at a specific point in time (e.g., user $u$ has to do action $a$ at 12am).

Finally, spatial operators allow the filtering of user actions with regards to the physical space, i.e., the condition is only fulfilled if the action takes place at a predefined location (e.g., user $u$ has to do action $a$ at pos = $(\text{latitude} = 1.2, \text{longitude} = 1.3)$).

**Joint Actions (CR21)** While constraints are subject to the actions and state of single entities (e.g., player, team, or system), joint actions are a necessary part to identify situations in which those entities did something together or against each other. In the literature, this aspect is usually described either as collaboration or competition game mechanisms (e.g., [109]). In this thesis the term Joint Actions is used as technical term to describe any situations where active entities (player, team, system, or environment) are doing something with or against each other in order to express game mechanics of conflict or cooperation.

**Randomness (CR22)** Randomness (Chance) might be used to introduce uncertainty and surprise to foster engagement [173]. Randomness can be based onto discrete or continuous probabilities and expresses the chance under which other conditions of a rule are fulfilled.

**4.2.2.3 RULE CONSEQUENCES (CR23)**

Finally, the consequences of rules are considered which are executed once arbitrary conditions of a rule are fulfilled. In these consequences, instances of the atomic L1 elements are generated which contributes to the evolution of the overall game’s state in general and the progress of entities therein in particular. For example, a rule’s consequence may give points, badges, generate notifications or narrative messages, or assign new missions to players or teams. Moreover, new events (internal or external) might be generated which triggers subsequent processing of rules.

**4.3 NON-FUNCTIONAL REQUIREMENTS**

While the functional requirements of the latter section refer to conceptual requirements which are generally independent of an underlying technology, the following requirements refer to some general characteristics and quality attributes that a supporting runtime system or environment has to provide from a non-functional perspective. In [57] a large list of such non-functional requirements for software systems is presented. Furthermore, in Section 2.2 general requirements and guiding design principles for enterprise software are discussed. In the following text, these general requirements are translated into specific quality attributes for gamification runtime systems. Here, only those requirements are considered which can be derived from the gamification development process (Chapter 3). Additional attributes which are not considered in the following text, are subject to future work.
4 Conceptual Requirements

Performance:
The feedback of the gamification should be immediate in accordance to, e.g., [136, p. 21]. Real-time can be either classified as hard, firm, or soft real-time. Hard real-time requires the processing time $t_p$ of an algorithm to be always smaller than the sampling period $T$, i.e., the duration between two events

$$t_p + t_o < T$$

(4.1)

with $t_o$ being the I/O overhead time [120]. In soft real-time, $t_p$ might be larger than $T$, however, the information’s usefulness diminishes with $t_p - T$.

This implies for a generic gamification runtime environment that the evaluation of rules in conjunction with the object state has to be processed at least in perceived (soft) real-time. The commonly accepted absolute threshold of 500ms for interactive systems has to be ensured by a gamification runtime system [139].

Flexibility:
As defined by [118, p. 25], this requirement is generally necessary in enterprise settings to allow for changes to the IS when the organization changes. This also has to apply to gamification technologies since the gamification concept may change continuously. Hence, the supporting runtime system has to be able to adopt new design demands and business requirements after the deployment and during the monitoring phase. For example, if the gamification expert decides to give users three points instead of five points for a particular user action, this has to be changeable without rebuilding or redeploying the IS.

Invasivity:
This attribute refers to the degree, how invasive the gamification solution has to be implemented into the host application. Non-invasivity fosters a decoupling of functionality from technology, i.e., the actual IS must be kept independent from short-term technology innovations as defined by [76, 118]. This attribute mainly influences the decisions of the provisioning phase.

Integrability:
Although invasivity is required to be minimized, it is argued that gamification is always invasive to some degree (e.g., on the user interface side, to record or capture user actions from the IS etc.), hence, zero invasivity cannot be assumed for practical and non-trivial scenarios. Therefore, the remaining integration work of the gamification aspect and the target IS should be supported by methods and tools to allow for fast and flexible integration. Henceforth, integrability refers to the degree, how easy it is to integrate the gamification solution into the IS within the implementation phase.

This requirement is, again, dictated by the need for decoupling of functionality and technology as postulated by [76, 118].

Reusability:
This requirement refers to the degree, how reusable the considered gamification solution is in terms of code that can be reused and, consequently, has not to be implemented in the target information system or the host application. This quality attribute might be for example measured in percentage of covered requirements or reused lines of code.
4.4 Systematization

As such, it directly influences all phases of the gamification process starting from the provisioning phase.

**Manageability (Security):**
Privacy is a key concern when it comes to gamification due to the high levels of transparency it enforces [93]. For example, each user action in the system might be tracked and, thus, can be potentially misused for various reasons. Although this might be of interest to the top-management, in many countries the collection of such fine-grained employee data is forbidden. For example, in German enterprises, work councils are allowed to reject the introduction of systems that collect too much data of its employees. Thus, gamification platforms have to provide security mechanisms to anonymize the user’s identity and past actions to some degree.

Therefore, the persisted gamification data must be *manageable* across the gamification’s life-cycle. For example, data has to be aggregated or anonymized after a particular time to ensure legal compliance in some countries or domains. Regarding the gamification process this requirement primarily concerns the monitoring phase and refers to the degree of how many options are available to manage the gamification data.

**Analyzeability:**
Although this requirement seems to be functional by nature, in this thesis the non-functional aspect is considered and, thus, refers to the degree of how much data is accessible to provide the desired insights into user behavior (monitoring phase). As shown later, this quality attribute strongly depends on the selected architecture type.

Furthermore, this requirement trade-offs with other non-functional requirements, e.g., security or performance. For example, in the above paragraph it has been argued that data needs to be anonymized and aggregated. On the one hand, this limits the possibilities for analyses and desired insight into the data on a very detailed level. On the other hand, aggregated and managed data can be processed faster, hence, influences runtime performance positively.

### 4.4 Systematization

This concludes the consideration of functional and non-functional requirements in this thesis. With regards to the former, a literature review has systematized and disambiguated typical mechanisms for gamification from prior research. Each concept has been described briefly and informally with regards to goals and assumptions and has been classified using Deterding’s layered taxonomy.

Furthermore, all concepts have been labeled with numbers which are used as references in the remaining text, for instance, to formalize the concepts in Chapter 5 and to compare them against related work in Chapter 6.

Finally, this chapter discussed a set of non-functional quality attributes and their trade-offs for gamification technologies in the context of the overall gamification process. These requirements constitute the foundations for Chapter 7 in which a novel runtime environment for gamification is proposed.

In the next chapter, each functional concept is mapped onto a formal language construct of the so-called Gamification Modeling Language.
5 GAML - CONCEPTUAL GAMIFICATION MODELING LANGUAGE

Based on the conceptual requirements of Chapter 4, in the next sections, a formal language called the Gamification Modeling Language (GaML) is introduced. For each identified requirement, the mapping into the formal language is presented.

5.1 DESIGN OBJECTIVES

From a conceptual point of view, general-purpose and domain-specific languages can be distinguished. Furthermore, formal computer languages can be designed for different kinds of quality attributes such as readability, writeability, reliability, expressiveness and costs [55].

Hereinafter, GaML is considered as Domain-specific language (DSL) defined as: “a computer programming language of limited expressiveness focused on a particular domain” [82]. In particular, this work aims at an external DSL, i.e., a language that is independent from the underlying technology [82]. As the name implies, such languages are focused on a particular domain (e.g., gamification) and are less expressive than general-purpose languages, i.e., they cannot be used to express arbitrary things but concepts from the target domain only.

They are typical useful when improved development productivity, communication with domain experts, a shift of the execution context, and the decoupling of the specification from the underlying computational model (e.g., imperative programming, rule-based systems, or state machines) is necessary [82].

Based on these preliminaries, the design goals and assumptions for GaML are presented in the following:

Syntax (DO1): GaML formalizes the gamification concepts of Chapter 4. Based on this formalism, a parser can decide, whether a GaML instance is well-formed according to the language's grammar or not.

1 In the following, GaML qualifies as a formal language in accordance with [175]. Hence, formal semantics are not considered herein.
Semantics (DO2): Besides syntax, static semantics further validate well-formed instances of GaML with respect to the semantics defined in the conceptual requirements.

Specificity: (DO3): GaML is designed to be a domain-specific and declarative language, i.e., instances of the language describe what has to be accomplished instead of how it is accomplished. This is consistent with the assumptions of DSLs.

Readability (DO4): GaML should be at least readable and understandable for domain experts with minor IT knowledge, e.g., consultants or gamification designers which is consistent with [82].

Writeability (DO5): GaML should be writable for IT experts. According to [82], DSLs might not be suitable to be autonomously written by domain experts. Therefore, this design objective requires domain experts to accomplish at least simple modifications based on existing well-formed GaML instances.

Compileability (DO6): GaML should be automatically compilable into runtime environments for gamification, i.e., GaML is processed through an indirect way and, thus, may increase development productivity in the implementation phase.

These defined design goals are also evaluated in a user study later on (Chapter 8).

5.2 SYNTAX AND META-MODEL

In this section, the grammar and meta-model of GaML is presented. Additionally, the mapping of conceptual requirements to the formal grammar is shown.

5.2.1 GENERAL ELEMENTS

A context-free grammar is defined as quadruple $G = (N, T, P, S)$ with $N$ being the set of non-terminals, $T$ the set of terminals, $P$ the set of productions with $P = N \times (N \cup T)^*$, and $S \in N$ being the start symbol. In the following, the respective elements of the grammar are visualized using syntax railroads where non-terminals $N$ are visualized with rounded boxes and terminals $T$ in rectangular boxes. Furthermore, standard terminals for strings, Boolean types or IDs are defined as follows using regular expressions:

- **ID**: `'^'?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*`
- **String**: `'"' ( '\\' ('b'|'t'|'n'|"""|"""|'\') | !('"'|"""|'\') )* '"' | "' ' ('\\' ('b'|'t'|'n'|"""|"""|'\') | !('\\'|"""|'\') )* "'`
- **INT**: `('0'..'9')+`
- **DOUBLE**: `('0'..'9')+.'('0'..'9')+`

---

2Hereinafter, static semantics refer to the semantic rules that can be checked at compile-time on the parse tree in the course of semantic analysis [175]. Hence, the consideration of formal semantics is not covered.

3Since existing solutions for gamification cannot fulfill all conceptual requirements, in Chapter 7 the design and method of a novel runtime environment for gamification is described. To validate this approach, the transformation from GaML into the proposed runtime environment is shown as well.
5.2 Syntax and Meta-Model

- **BOOLEAN**: 'true' | 'false'
- **DATE**: "'" ('0'..'9') ('0'..'9') ('0'..'9') ('0'..'9') '-' ('0'|'1') ('0'..'9') '-' ('0'..'3') ('0'..'9') ('T' ('0'..'2') ('0'..'9') ':' ('0'..'5')('0'..'9')? (':'.('0'..'5')('0'..'9') ('T' ('0'..'2') ('0'..'9') ':' ('0'..'5') ('0'..'9')?))? "'

whereas standard semantics of the special symbols apply, i.e., ? determines an optional element, ! a negation operator, + a one to many and * a zero to many relationship of the structure surrounded by rounded brackets. In particular, the String definition allows any character sequence within single or double quotes except typical control characters, e.g., \n, \b, or \t for newline, backspace, or tabulator respectively.

In the following text, each conceptual requirement (CR#) is mapped to one or more syntactical elements of GaML. Each syntax element is, again, assigned a number (E#) which is used as reference to assess related work or to describe the code generation into runtime concepts later.

**Space (E1)** GaML’s grammar begins at the start symbol $S = Model$ which produces the terminal concept followed by an ID. Within the concept, multiple gamification elements (Element) might be defined, whereas also the empty concept is allowed (Figure 5.1). Hereby, Element reflects all tangible concepts which have been identified on L1 as atomic elements in Chapter 4. Therefore, this element maps the conceptual requirement CR11.

**Game Level (E2)** A game level is an optional element to divide the entire space of the gamification concept into smaller parts. Therefore, this construct contains the same subelements as the global concept element by referencing the non-terminal Element recursively in its body again (Figure 5.2). This construct maps requirement CR12. Besides the division of the overall gamification concept into smaller parts, further attributes might be declared. This includes at least the level’s name and completion criterion, i.e., when the game level is finished (see Section 5.2.2), as mandatory attributes. Optional parameters determine if all uncompleted missions of this game level are set to “completed” when the game level’s completion criterion becomes true or which subsequent level follows. It is important to note that in this thesis exactly one game level follows upon completion of another level.

**Event (E3)** The declaration of events (CR8) within the gamification concept is introduced by the event’s class (which produces either userevent, externalevent, or internealevent reflecting events’ subtypes), followed by the event’s ID and an optional event body (Figure A.7). Within the event’s body the optional attributes $\text{inverseName}$ and $\text{data}$ can be defined. First, the data attribute might be used to statically define properties describing the event further including their data types. These properties are statically typed since GaML is not supposed to be executed directly, but indirectly through intermediate languages (see Chapter 7). Through this indirection, runtime type checking is omitted by default. Second, the attribute inverseName declares the name of the inverse event and requires exactly one reference to a data property to define which of the properties joins the event with its inverse.

4 All syntax railroads are documented comprehensively in Appendix A.
Achievement (E4)  In conjunction with [90], the definition of an achievement includes at least a name, visual badge, and description of the goal for this achievement (CR2). The formal definition of an achievement is, therefore, introduced by the terminal badge, followed by the achievement’s ID (Figure A.8). Within curly brackets the concept is further described by its display name, an optional description, and image. Furthermore, the achievement is declared as visible or hidden, i.e., if the achievement’s goal is visible to the user at runtime or not. Finally, an optional condition, i.e., either none or exactly one condition, for getting the achievement can be described formally (see Section 5.2.2).

Point Category (E5)  Likewise, the definition of a point category (CR1) is started with the terminal point and the point’s ID (Figure A.9). Within the concept’s body, at least the point’s display name has to be defined. Additionally, optional attributes comprise an abbreviation and description of the point category in question. Furthermore, the type of the point category can be specified (E5a; Figure A.9) and the point can be marked as default, i.e., the point is used as the primary metric later. Finally, multiple rules with different conditions and consequences (see Sections 5.2.2 and 5.2.3) can be specified signifying the situations under which instances of this point category can be achieved.
5.2 Syntax and Meta-Model

**GameLevel**

![Diagram of GameLevel element](image)

**Mission (E6)** Within the mission element, arbitrary formal conditions and consequences (i.e., rules) can be defined which can be informally described using a display name and description (Figure A.10). The syntax requires an author to define at least one rule. Multiple rules can be connected through Boolean operators whereas those refer to the overall Boolean state of the individual conditions. If no operator is given, implicitly logical “and” (higher precedence as logical “or”) is assumed. Once the Boolean statement is fulfilled, the mission is considered as completed.

In correspondence with Chapter 7, authors may define an optional available clause in order to express the conditions when this mission becomes available to a player or team (E6a). It is important to note that arbitrary conditions can be defined here (see Section 5.2.2). If no availability condition is specified, the corresponding mission is considered as a starting mission and is assigned to the user on his first interaction.

Upon assignment, the attribute initiatedBy (E6b) determines whether the mission is assigned automatically (rule driven) or if the user has to explicitly agree to take the offered mission (user driven). Since this is an optional attribute, by default an automatic mission assignment is assumed. Finally, this concept comprises multiple rules, i.e., with arbitrary conditions and consequences for players or teams. Only if all rules are fulfilled, the mission in question is considered as completed.

This element reflects the discussed requirements of CR6.

**Skill (E7)** Reflecting requirement CR4, Figure A.11 presents the formal definition of the concept skill introduced by the same-named terminal followed by the skill’s ID,
display name, description, image, and the benefit of the skill in natural language. Furthermore, the skill has to be linked to a specific point category expressing the various levels for skills. Note that the validation whether the referenced point category is of type `SKILLPOINT` is subject to the static semantics of the language as described in Section 5.3.

Moreover, this element may define a particular availability period. This declares the skill as an active one giving the promised benefit only for the specified time duration. If no such duration time is specified, the skill is considered as passive, i.e., the benefit of the skill applies every time.

**Level (E8)** Conceptual requirement CR14 is mapped through the level element which contains the level's name and the threshold with reference to a previously defined point category (Figure A.12). Since levels can be defined for different types of point categories, there is no check with regards to the point type semantics on levels. Optionally, levels may specify a reference to an image.

**Good (E9)** The definition for goods (CR3) is introduced by the keyword `good` followed by its ID and at least the good’s name, description, and image. The good’s type, i.e., if it is a virtual item or a real good might be optionally defined (E9a) whereas a real good is assumed by default, if this attribute is not specified (Figure A.13). Furthermore, the good might be associated with an internal price (i.e., referencing a point category with type being redeemable) to generate a marketplace at runtime automatically.

**Role (E10)** Roles describe various different character roles for the gamification (CR5). Each role has at least an ID, display name, image, and optionally a description (Figure A.14). Moreover, roles may reference specific skills, i.e., these skills can be only earned and used by users having this particular role. These skills are referenced through their ID as defined by element `E7`.

**Leaderboard (E11)** Although the content of the various leaderboards is assembled at runtime, the specification, i.e., aggregation function, can be already defined at design-time in order to map CR17. As shown in Figure A.15, the leaderboard is identified through an ID and further specified by name, associated point category, the aggregation function, the order (ascending, descending), and the target entity (player, team) either for the graphical leaderboard generation at runtime or the determination of leading users within the various conditional blocks described above.

### 5.2.2 CONDITION ELEMENTS

Besides the specification of the L1 elements, conditions for achieving the various elements can be defined. For example, the constructs for badges (E4), points (E5), missions (E6), or skills (E7) explicitly allow the definition of single or multiple conditions within their defining body. The general syntax for conditions is shown in Figure 5.3.

In general, each condition is introduced by the keyword `when`. Afterwards, the author has to decide if the condition applies to an individual (i.e., `player`) or a group of individuals (i.e., `team`; CR19). Both entities can be assigned a variable (EntityVariable) which is used for identification of an instance later on. Despite the case, inside the subsequent block, optional validity start and end dates can be specified. Furthermore, player- or team-specific conditions can be formulated within the non-terminals `PlayerOr`
When

\[ \text{PlayerEntity} \{ \text{validFrom} = \text{DATE}, \text{validTo} = \text{DATE} \} \]

\[ \text{TeamEntity} \{ \text{validFrom} = \text{DATE}, \text{validTo} = \text{DATE} \} \]

\[ \text{PlayerEntity} \{ \text{player} \} \]

\[ \text{TeamEntity} \{ \text{team} \} \]

\[ \text{EntityVariable} \{ \text{ID} \} \]

Figure 5.3: General condition syntax
or TeamOr (Figure 5.3). It is notable that some conditional elements can be used in both contexts, while others are specific to the selected entity. Hence, the next subsections describe elements which are independent first. Subsequently, player- and team-specific conditions are explained in more detail.

5.2.2.1 ENTITY-INDEPENDENT CONDITIONS

This subsection subsumes all conditions which are defined and available for both entities, i.e., players and teams. Hence, an author may use all the constructs explained below in both contexts.

**Boolean Constraints (E12a):** Both entities have in common that conditions can be arbitrarily correlated by Boolean operators. Figure 5.4 presents an excerpt of the grammar for Boolean operations within the player entity which implements requirement CR20a.

The non-terminal PlayerOr produces arbitrary sequences of the non-terminal PlayerAnd followed optionally by terminal or. The same applies for the non-terminal PlayerAnd, except that it produces the non-terminal PlayerTopElem and terminal and etc.

Eventually, this chain results in the non-terminal PlayerCondition which references general conditions subsumed under the non-terminal Condition. These conditions are considered in the following text.

Furthermore, there are four player-specific conditions which are described in Subsection 5.2.2.2.

It is important to note that the grammar of Figure 5.4 applies to the team entity as well (Figure A.19), except that it produces the non-terminal TeamCondition in the end which defines one team-specific term as shown in Figure 5.5. This team-specific feature is covered in Subsection 5.2.2.3.

**General Conditions (E12b-E12e)** In this paragraph all conditional elements are subsumed that refer to the context of players and teams, thus, realizing requirement CR13. This applies to the condition of tangible concepts such as badges (E12b), levels (E12c), goods (E12d), and missions (E12e). Defining these conditions means that an entity has to hold at least one instance of the concept in question, e.g., user \( u \) has to own badge \( b_1 \) or has mission \( m_1 \) assigned currently. The syntax is given in Figures A.23 and A.24. It is important to note that the language provides singular and plural terminals for referencing either exactly one or multiple instances of the same concept respectively. For example, a condition may refer to exactly one badge (e.g., has badge \( a \)) or multiple ones (e.g., has badges \{a, b\}).

**Skill Condition (E12f)** Although the skill condition is very similar to the general references and, therefore, is also a realization of CR13. However, this concept additionally includes an optional skill level that has to hold in order to evaluate this condition to true, i.e., the player or team has a particular skill at the given level (Figure A.25). Furthermore, the condition may check if the skill is activated for the enclosing entity.

**Point Condition (E12g)** The point condition evaluates true if the entity in question has the defined amount of points and, in turn, belongs to the realization of CR13. The syntax allows to specify the point type and aggregation method compared to a numeric expression (Figure A.26).
5.2 Syntax and Meta-Model

**Figure 5.4: Boolean constraint and conditions syntax**

**Luck Condition (E12h)** The luck condition evaluates true, when a (pseudo-)random number is smaller than the given numeric expression (Figure A.27). Here, $\text{NUMEXPR} \in [0, 1]$ has to hold for all specifications. This condition explicitly maps CR22.

**Location Condition (E12i)** The location condition is used to determine whether the issuing entity is at the specified geo-location or not (Figure A.28) and, thus, implements CR20d.
Leader Condition (E12j) The leader condition determines the best \( n \) entities according to the specified parameters, i.e., starting (from) and end (to) position whereas the former is an optional attribute. If no starting value is given, this refers to the top player or team within the condition. Furthermore, aggregation function and point category are implicitly derived from the referenced leaderboard specification (Figure A.29).

5.2.2.2 PLAYER-SPECIFIC CONDITIONS

After describing conditions that apply to both, teams and players, the following text describes player-specific conditions as presented in Figure 5.4.

Role Condition (E12k) In this thesis, only players or users may own a pre-defined role. Therefore, this condition element as shown in Figure A.30 examines, if a player has the required role assigned. Thus, this element implements conceptual requirement CR5 and CR13.

Event Condition (E12l-E12p) As presented in Figure A.21, conditions referring to an event are introduced by the terminal \texttt{did} followed by an optional variable declaration and a reference to exactly one previously declared event type (Section 5.2.1). Separated by commas, the event can be filtered based on constraints (E12m) on the event's predefined properties. This includes numeric constraints as well as string or Boolean constraints, thus, reflecting CR20b. Additionally, not only simple data types can be compared with each other but other events might be referenced based on a reference operator (E12n). This can lead to arbitrary join conditions that are required to reflect collaborative and competitive scenarios upon event processing (CR21). Furthermore, the event can be temporally (E12o→CR20c) correlated with other events using 13 different temporal operators [6]. Besides event correlation, these temporal operators might be also used to correlate the event in question with a fixed timestamp in ISO format, i.e., if the event in question happens before, after, or at a specific timestamp.

Further temporal aspects within the condition comprise continuous time windows with numeric aggregations (e.g., count, sum, average), and a special \texttt{lastsFor} operator. (E12p→CR20c). The latter might be used to express scenarios, where the event in question has to reside at least for the given time in the system before the consequence is issued. This is for example necessary to prevent cheating scenarios.

Finally, this concept refers not only to events of the same player but also events from other players might be taken into account. For this purpose, there is the sentence another player \texttt{did} \texttt{<EventDef>} available which might be used to express that another player did some particular event. Hereby, the same syntax as described above is used to filter and correlate events.
5.2 Syntax and Meta-Model

**Team Belongingness (E12q)** The last player-specific element is used to identify, if a player belongs to a particular team which itself is in a particular state. Figure A.18 shows the corresponding syntax. Since the player can be in multiple teams, this statement may evaluate true for all combinations of the player and his teams. In order to filter the result set further, the `which` clause might be used to determine only the teams that are in a particular state. In this case, all team-specific conditions produced by the non-terminal TeamOr can be used again. Additionally, all teams that the player belongs to can be bound to a variable. This can be used, for example, to give the entire team a reward.

5.2.2.3 TEAM-SPECIFIC CONDITION

**Player Belongingness (E12r)** Besides general and player-specific conditions, there is one condition specific to the entity as presented in Figure 5.5. This special statement might be used to check, if any player of the team is in a particular state. As such, this condition is the counterpart to the team belongingness condition (E12q) available within the player conditions. Therefore, all players can be referenced who belong to that team. The matching set of players can be further pruned by providing player conditions over the non-terminal PlayerOr as presented in Figure 5.4.

5.2.3 CONSEQUENCE ELEMENTS

After some condition has been fulfilled, corresponding consequences are executed. Those can be of the following types: internal events (E13a), points (E13b), notifications (E13c), narrations (E13d), and so-called general consequences where badge (E13e), good (E13f), mission (E13g), and skill (E13h) consequences are subsumed as shown in Figure 5.6. All of these language elements can be mapped to CR23. In the following, each element is presented in more detail.

\[
\text{Then} \quad \{ \text{Consequence} \} \quad \{ \text{Consequence} \}
\]

**Figure 5.6: General consequence element**
Internal Events (E13a): One possible consequence is the creation or deletion of an internal event which might be necessary to execute subsequent rules. The syntax for this element is given in Figure A.33. This element is introduced with either the terminal create or delete followed by a reference to one of the previously defined events. It is important to note, that this reference is valid on internal events only since the GaML author should be able to model these events, i.e., they cannot be issued directly from a user. Static semantics are used later on top of this meta-model to ensure integrity. Arbitrary properties can be defined for this event whereas numeric expressions, references to other events in the condition clause, or Boolean and string types might be assigned.

Finally, there is an optional clause specifying that the event is given to a particular entity. For this purpose, the variable bindings from the condition clause are utilized. If no variable binding is specified then automatically the utmost entity of the condition is used as presented in the section for translational semantics (Section 7.6.2.1).

Points (E13b): Points might be added to or removed from a specified entity which fulfilled the corresponding condition. Furthermore, points might be also set to a specific amount. The syntax for point consequences is shown in Figure A.34. Again referential integrity is ensured using static semantics as shown later.

Notification (E13c): Explicit notifications might be assigned upon fulfilled conditions. There is a short and a long form for this concept. The short form only requires a string and, optionally, the receiver for the notification. Additionally, the long form requires the specification of a title and, optionally, an image and notification type (corrective or information). Also this language element addresses conceptual requirement CR7, CR10, and partially CR18. The syntax is shown in Figure A.35.

Narration (E13d): Narrative messages are also possible consequences upon completion of conditions. The syntax for narration is presented in Figure A.36. The element requires at least a name and text for the narrative message. Optional attributes comprise links to audio, video, or image files which are presented or played back as consequence of the condition.

Badges (E13e), Items (E13f), Missions (E13g), Skills (E13h): Mission, badge, item, and skill consequences are treated equally as presented in Figure A.37 and are subsumed under the non-terminal GeneralRefcons. Each element references one previously defined concept through its ID. This means that an instance of the referenced concept is assigned to or removed from the respective entity.

5.3 STATIC SEMANTICS

Besides syntax, static semantics are defined for the language, which are validated on the syntax tree of well-formed GaML instances. With static semantics several additional aspects such as integrity, type safety or logical soundness can be validated. Semantics are defined using logical expressions, e.g.,

\[ \forall a, b: \text{Badge}(a) \land \text{Badge}(b) \land a \neq b \Rightarrow \text{Name}(a) \neq \text{Name}(b) \]

using the Object Constraint Language (OCL).

More precisely, static semantics are defined for the following aspects:
5.3 Static Semantics

- Uniqueness of element IDs and attributes (e.g., all points must have unique IDs).
- Referential integrity for explicit links between element definitions (e.g., the definition of the `level` elements refers to some particular point category).
- Minimum and maximum cardinalities of children elements or properties (e.g., each GaML instance must have at least one element defined).
- Cyclic dependencies in the specific mission graph (e.g., none of a mission’s children should point back to the mission in question).
- Type checking in numeric expressions, especially when variables are present (e.g., variables of type String must not be used in numeric expressions).
- Use of correct point types (e.g., only redeemable point metrics have to be used as price definition within goods)

Listing 5.1 shows, for example, the OCL constraint for the uniqueness of all event IDs. Note that the attribute `name` in the examples below denotes the terminals of the grammar.

```ocl
context Type
c
inv UniqueEventID:
event.allInstances()->forAll(a, b| a <> b implies a.name <> b.name);
```

Listing 5.1: Constraint for uniqueness of IDs

Listing 5.2 presents example constraints to check for the correct point type usage. For example, within the good element, the invariant checks whether the referenced point is of type redeemable and raises an error, if it is not. As another example, the integrity in a skill definition is checked, i.e., if the referenced point is of type skill.

```ocl
context Good
c
inv isRedeemablePointType:
point.allInstances()->exists(p|p=point and p.type.name='REDEEMABLE')

c
context Skill
c
inv isSkillPointType:
point.allInstances()->exists(p|p=point and p.type.name='SKILL')
```

Listing 5.2: Constraints for point type semantics

Listing 5.3 shows an example for checking the point types with regards to point operations semantically, e.g., that points can be only removed for point type redeemable.

```ocl
context PointRefCons
c
inv RemoveImpliesRedeemable:
(self.action='remove' and self.point<>null) implies self.point.type.name='REDEEMABLE';
```

Listing 5.3: Constraint for point semantics

Listing 5.4 shows the definition of custom OCL operations to checks if a mission graph is free of cycles. For this purposes the operations `traverseBoolExp` and `collectMissions` are used to traverse the mission graph along the available clauses (root.condition) and to recursively collect all parent missions of the current one. Based on the resulting set, the invariant specified that the mission in question must not be contained whereas this has to hold for all missions within the entire graph.
context Mission

inv AcyclicPrecondition:
    Mission::requiredMissions(self)->excludes(self);

operation requiredMissions(root: Mission): Set {
    body: if (root.condition <> null)
        then Mission::traverseBoolExp(root.condition.boolExp)
        else Set()
    endif;
}

operation traverseBoolExp(boolExp: Expression): Set {
    body: if (not boolExp.atom.oclIsUndefined())
        then Mission::collectMissions(boolExp.atom)
        else Set(boolExp.left, boolExp.right, boolExp.'not')
            ->reject(bExp | bExp.oclIsUndefined())
            ->collect(bExp | Mission::traverseBoolExp(bExp))
            ->flatten()
            ->asSet()
    endif;
}

operation collectMissions(condition: Condition): Set {
    body: if (condition.condition.oclIsTypeOf(Expression))
        then Mission::traverseBoolExp(condition.condition)
        else if (condition.condition.oclIsTypeOf(MissionItem))
            then let missionItem: MissionItem = condition.condition.
                oclAsType(MissionItem)
                in Set{ missionItem.mission }
                ->union( Mission::traverseBoolExp(missionItem.mission.
                    condition.boolExp) )
            else if (condition.condition.oclIsTypeOf(MissionRefs))
                then let missionRefs : MissionRefs =
                    condition.condition.oclAsType(MissionRefs)
                    in missionRefs.missions.mission
                    ->asSet()
                    ->union(missionRefs.missions.mission
                    ->collect(m | Mission::traverseBoolExp(m.
                        condition.boolExp))
                    ->flatten())
                else Set()
            endif
        endif
    endif;
}

Listing 5.4: Constraint and operations for acyclic missions
Finally, type checks are performed on the properties of events, i.e., within a numeric expression it is checked whether the referenced properties of events are declared as numerical types as shown in Listing 5.5.

```java
context NUMEXPR
inv NumeralOperands: if(self.op1.refOp->notEmpty())
then
  let
    dType : String = self.op1.refOp.var.dType.name
  in
    Set{'Number', 'Decimal'}->includes(dType)
else
  true
endif;
```

Listing 5.5: Constraint for type checks in numeric expressions

Some of these constraints may result either in errors (e.g., type checks) or warnings (e.g., point types).

In this thesis, dynamic semantics are not defined since it is assumed that GaML is not directly executable. However, in Chapter 7, GaML's semantics are at least described as translational semantics [112], i.e., GaML instances are translated into an existing language with pre-defined computational model (e.g., Drools Rule Language). The definition of operational, axiomatic, or denotational semantics are subject to future work (Chapter 9).

### 5.4 SYSTEMATIZATION

In this chapter, GaML and its syntax and static semantics has been introduced. Table 5.1 gives an overview of all language elements and their relationship to at least one of the conceptual requirement. It is important to note that not all conceptual requirements are mapped by the language as they have been identified as runtime concepts. For instance, the marketplace concept has been omitted as it is assumed that all virtual and real goods can be traded within the bounds of the monetary system. Therefore, no additional information is required at design-time. Other concepts defined as runtime concepts are necessary to be referenced implicitly, i.e., they are reflected within the gamification’s logic only. For example, the concepts of avatars or teams are not expressed directly in GaML as they are considered primarily as runtime concepts. However, within GaML rules, authors may decide if particular conditions apply to players or teams. These implicit mappings are given in rounded brackets of Table 5.1. The underlying runtime environment has to support these concepts in addition to the ones that can be already defined at design-time.

<table>
<thead>
<tr>
<th>CR#</th>
<th>Mapped by</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1 (Points)</td>
<td>E5, E5a</td>
<td></td>
</tr>
<tr>
<td>CR2 (Achievements)</td>
<td>E4</td>
<td></td>
</tr>
<tr>
<td>CR3 (Goods)</td>
<td>E9, E9a</td>
<td></td>
</tr>
<tr>
<td>CR4 (Skills)</td>
<td>E7</td>
<td></td>
</tr>
<tr>
<td>CR5 (Roles)</td>
<td>E10</td>
<td></td>
</tr>
<tr>
<td>CR6 (Missions)</td>
<td>E6, E6a, E6b</td>
<td></td>
</tr>
<tr>
<td>CR#</td>
<td>Mapped by</td>
<td>Comment</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>CR7 (Feedback)</td>
<td>E13d, E13c</td>
<td></td>
</tr>
<tr>
<td>CR8 (Events)</td>
<td>E3</td>
<td></td>
</tr>
<tr>
<td>CR9 (Narrative Context)</td>
<td>(E13c)</td>
<td>Runtime concept</td>
</tr>
<tr>
<td>CR10 (Notifications)</td>
<td>E13c</td>
<td></td>
</tr>
<tr>
<td>CR11 (Space)</td>
<td>E1</td>
<td></td>
</tr>
<tr>
<td>CR12 (Game Level)</td>
<td>E2</td>
<td></td>
</tr>
<tr>
<td>CR13 (Context)</td>
<td>(E12b), (E12c), (E12d), (E12e), (E12f), (E12g), (E12j), (E12k), (E12q), (E12r)</td>
<td>Runtime concept</td>
</tr>
<tr>
<td>CR14 (Avatar Levels)</td>
<td>E8</td>
<td></td>
</tr>
<tr>
<td>CR15 (Avatar)</td>
<td>(E12a)</td>
<td>Runtime concept</td>
</tr>
<tr>
<td>CR16 (Marketplace)</td>
<td>-</td>
<td>Runtime concept</td>
</tr>
<tr>
<td>CR17 (Leaderboard)</td>
<td>E11</td>
<td></td>
</tr>
<tr>
<td>CR18 (Communication System)</td>
<td>(E13d), (E13c)</td>
<td>Runtime concept</td>
</tr>
<tr>
<td>CR19 (Team)</td>
<td>(E12q), (E12r)</td>
<td>Runtime concept</td>
</tr>
<tr>
<td>CR20a (Boolean)</td>
<td>E12a</td>
<td></td>
</tr>
<tr>
<td>CR20b (Event)</td>
<td>E12l, E12m</td>
<td></td>
</tr>
<tr>
<td>CR20c (Temporal)</td>
<td>E12o, E12p</td>
<td></td>
</tr>
<tr>
<td>CR20d (Location)</td>
<td>E12i</td>
<td></td>
</tr>
<tr>
<td>CR21 (Joint Actions)</td>
<td>E12n</td>
<td></td>
</tr>
<tr>
<td>CR22 (Randomness)</td>
<td>E12h</td>
<td></td>
</tr>
<tr>
<td>CR23 (Consequences)</td>
<td>E13a-E13h</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Mapping of conceptual requirements and language specification

In the following chapter, these specific language elements are used to assess existing runtime environments for gamification. Additionally, for each language element the transformation into the generic runtime system developed in this thesis is shown as well.
6 RELATED WORK

This chapter presents related work in the domain of gamification. More specifically, for each contribution of this thesis a separate section outlines the state-of-the-art as well as the addressed gaps. First, existing languages for games and gamification are analyzed. Second, runtime environments for gamification are discussed with regards to the requirements presented in Chapters 4 and 5.

6.1 LANGUAGES

The state-of-the-art in the field of gamification currently lacks a proper modeling language for gamification concepts that supports domain, gamification, and IT experts in the design phase of gamification concepts. Therefore, some researchers started using standard mathematical descriptions to describe their gamification concept (e.g., [29]) in a formal way. However, a proper language is missing.

Nevertheless, there are other established game languages and modeling approaches. These approaches are analyzed below and compared with GaML’s design goals (Section 5.1). Based on this analysis, it is shown that none of the existing approaches fully suffices all the requirements of gamification. Based on this observation it is concluded that the proposed GaML not only serves the purpose of formalizing the identified gamification concepts and game mechanics but might be used to fill theoretical gaps in the state-of-the-art.

6.1.1 GAME DESCRIPTION LANGUAGE

The Game Description Language (GDL) is a formal language to describe emergent games for general game playing. In this domain, autonomous agents (virtual players) try to compete against each other without any information but the description of the game given in GDL. The language comprises eight primitives to express the rules of arbitrary emergent games such as chess or tic-tac-toe. In general, the keywords allow to describe what a player \( \text{role}(R) \) and what the initial state of the game is \( \text{init}(F) \). Additional elements describe which moves are allowed \( \text{legal}(R,M) \), which moves are taken by the player \( \text{does}(R,M) \), and which state terminates the game \( \text{terminal} \). The language can be extended to typical concepts of board and card games such as turns, places, or tokens which can be used in the context of the keywords shown above. For example, Listing 6.1 shows an excerpt of the game Tic-Tac-Toe written in GDL. This
example comprises the definition of 9 initial fluents (e.g., the places where token might be placed by players), legal moves (e.g., if player \( W \) is in control and cell\((X,Y)\) is its initial state, then \( W \) is allowed to mark the cell in question), update rules (e.g., the cell is marked with \( x \) or \( o \) if the current player marks the corresponding cell, if, however, the cell is already marked, it retains the mark), and terminals for winning states (e.g., when a row, diagonal, or column is drawn by one of the players).

```
1 init(cell(1,1,b)).
2 init(cell(1,2,b)).
3 init(cell(1,3,b)).
4 init(cell(2,1,b)).
5 init(cell(2,2,b)).
6 init(cell(2,3,b)).
7 init(cell(3,1,b)).
8 init(cell(3,2,b)).
9 init(cell(3,3,b)).
10 init(control(white)).
11
12 legal(W,mark(X,Y)) :- true(cell(X,Y,b)) & true(control(W))
13 legal(white,noop) :- true(control(black))
14 legal(black,noop) :- true(control(white))
15
16 next(cell(M,N,x)) :- does(white,mark(M,N)) & true(cell(M,N,b))
17 next(cell(M,N,o)) :- does(black,mark(M,N)) & true(cell(M,N,b))
18 next(cell(M,N,W)) :- true(cell(M,N,W)) & distinct(W,b)
19 next(cell(M,N,b)) :- does(W,mark(J,K)) & true(cell(M,N,b)) & distinct(W,J)
20 next(cell(M,N,b)) :- does(W,mark(J,K)) & true(cell(M,N,b)) & distinct(W,K)
21 next(control(white)) :- true(control(black))
22 next(control(black)) :- true(control(white))
23
24 terminal :- line(W)
25 line(X) :- row(M,X)
26 line(X) :- column(M,X)
27 line(X) :- diagonal(X)
28 terminal :- ~open
29
30 open :- true(cell(M,N,b))
```

Listing 6.1: Excerpt of TicTacToe in game description language

Although very powerful and expressive, the goals of this language completely differ from the assumptions of this thesis as stated in Chapters 4 and 5. For instance, GDL assumes that players act on complete information excluding only the strategies of the other players. This is consistent with the observation that GDL is a language to primarily model emergent games. Within gamification, however, the rule set is exposed step-by-step to the user, i.e., users act on incomplete information all the time.

Based on the example, it is argued that the syntax and semantics of the GDL are very technical and based on predicate calculus making it arguably hard for non-technical experts to read or write their own gamification concepts. Furthermore, expressing progression games, which are of iterative nature, in GDL, which requires recursive descriptions, results in a cumbersome translation activity that can only be achieved by high-skilled personnel. However, this is not consistent with the assumption of this thesis and contradicts design goals \textbf{DO4} and \textbf{DO5}.

Furthermore, the GDL allows to define fluents and states freely, i.e., there is no pre-defined meta-model for games or gamification concepts with well-defined and established semantics available. However, it is argued that this is absolutely necessary to allow efficient discussion or implementations. Hence, \textit{GaML} is a proposal to provide such a semantic model which, in turn, validates the corresponding taxonomy for
the domain of gamification. This goal is not accomplished by the GDL and, therefore, contradicts design goals \textbf{DO1} and \textbf{DO2}.

Additionally, valid GDL instances or descriptions cannot be executed or translated automatically into arbitrary runtime environments for gamification but are interpreted by autonomous solvers. Hence, design goal \textbf{DO6} is not fulfilled by the GDL given the prior assumptions.

\subsection*{6.1.2 MACHINATIONS}

In [68] a visual modeling framework called \textit{Machinations} is proposed. This framework allows game designers to model game mechanics or parts thereof in a visual manner. For example, Figure 6.1 shows an exemplary positive feedback loop for the game \textit{Monopoly}. Based on such descriptions, simulations can be run to study emergence effects, e.g., if feedback loops are correctly balanced. Therefore, the framework might be used by game designers to study and validate their game mechanics on a theoretical level even before its final implementation.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure6.1.png}
\caption{Example for Machinations framework [68]}
\end{figure}

Although the framework is well-suited to study emergence effects of specific game mechanics, it cannot be used to express entire gamification concepts directly.

First, important mechanics for the progression aspect of gamification are missing (\textbf{DO1} and \textbf{DO2}). In particular, the framework enables the modeling of missions including abstract sub tasks and their interdependencies only. However, modeling the complex rules that determine the tasks as well as the progression between these tasks including user feedback is not possible. For example, it is not possible to model joint actions of users or temporal constraints that are necessary to complete some specific task. Additionally, it is not foreseen to explicitly model the flow of feedback to the user.

Second, due to its visual representation modeling all structures and functions of a holistic concept is limited to some degree. A simple but already complex looking example is given in Figure 6.2. Therefore, only parts of the entire game or gamification concept can be modeled clearly and unambiguously at one point in time.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure6.2.png}
\caption{Example for Machinations framework [68]}
\end{figure}

Based on Figure 6.2, it is further argued that design goals \textbf{DO3} and \textbf{DO4} are only partially fulfilled, as the syntax and semantics of the graphical elements is not intuitively clear to domain, gamification, or IT experts. Hence, prior trainings are necessary to let others understand the impact of the solutions.

Third, the language can be used for simulation purposes only to study emergence effect, i.e., the language cannot be directly compiled to an underlying technical framework to put the modeled game mechanics into host environments such as games or host applications for gamification (\textbf{DO6}).
Fourth, the proposed language contains control structures that force the designer to care about modeling how the game mechanics are implemented (e.g., feedback loops through repetition elements). Thus, it does not fully comply with design objective **DO3**, although it is acknowledged that most modeling elements are of descriptive nature.

### 6.1.3 SERIOUS GAME STRUCTURE AND LOGIC MODELING LANGUAGE

[188] introduce the Serious Game Structure and Logic Modeling Language (GLiSMo) as a DSL for serious games. The meta-model includes concepts such as **objects**, **characters**, **acts**, or **scenes**. Furthermore, the language comprises **actions**, **tasks**, and **assessments** to represent the game’s logic. There are different types of tasks, namely **multiple-choice** and **drag-and-drop** where the former is used to challenge the user with quizzes and the latter to combine graphical objects in predefined shapes. Assessments are then used to measure the correctness of the user’s solution.

However, the concept has major drawbacks with regards to gamification. First, the task types are fixed, i.e., there are only multiple-choice and drag-and-drop quizzes available. Hence, there is no possibility to freely design rules with conditions and consequences.

Second, the language only comprises notifications, i.e., other types of feedback or visual elements are not foreseen (Chapter 4). Therefore, it is argued that the language
does not support design goal \textbf{DO1} adequately. Since, the authors do not report on semantics, design goal \textbf{DO2} is not fulfilled as well.

Third, concepts such as acts or scenes have not been identified as requirement for gamification and are, thus, out of scope for the proposal of \textit{GaML}. Moreover, the currently reported status of the language mixes implementation details (e.g., GUIManager or AudioManager) with general concepts (e.g., Characters or Scenes). Therefore, it remains unclear which concepts constitute the language. Finally, the language is of descriptive character only, i.e., it is not translatable into execution environments automatically (\textbf{DO6}).

\subsection*{6.1.4 OTHERS}

Surveying the literature yields further formal languages. However, in contrast to the ones described above, those languages are tailored to specific aspects of games rather than gamification and cannot be reused for the purposes of this thesis. This includes the Card Game Description Language (CGDL) [79] tailored to the simulation and derivation of novel card games, the Strategy Game Description Language (SGDL) [132] specific to the description of emergent strategy games, and the Video Game Description Language (VGDL) [72] which might be used to describe and generate simple video games. In the following, the CGDL is used as an example to show that there are no overlaps with the design objectives of \textit{GaML}. It is argued that this description also applies to all other identified languages.

The approach by [79] presents the Card Game Description Language (CGDL), which allows for the modeling of card games, i.e., specific emergent games, in a well-defined language. Besides syntax, the authors describe axioms that have to hold for all instances of the language.

In general, each instance following the meta-model as well as the axioms represents a valid card game by definition. Within a randomized simulation, the authors show that well-known games such as \textit{UNO} or \textit{Poker} converge against predefined quality criteria such as number of turns or draws. Based on these findings the authors propose to automatically derive novel card games by simulation only.

However, in general this language does not fit to premises of this thesis. First, the language is limited to the description of card games only. Consequently, the language requires the definition of at least one winning state. Furthermore, the meta-model foresees concepts such as stages, turns, tokens, and places which are not relevant as gamification concepts. Thus, the language contradicts with design goal \textbf{DO1}.

Second, the language’s semantics are not of general nature but tied to the semantics of card games. For example, a player’s turn is over when he or she is done, next, or out. Hence, design goal \textbf{DO2} is not fulfilled.

Third, also the CGDL is based on predicate calculus and, thus, hardly understandable or modifiable as postulated in this thesis (contradiction with design goals \textbf{DO4} and \textbf{DO5}).

Due to the described limitations, the language is consequently not translatable to gamification environments (\textbf{DO6}).

\subsection*{6.1.5 SYSTEMATIZATION}

The key features of the analyzed languages are presented in Table 6.1. In general, it is notable that the majority of existing languages focus on emergent games (namely GDL, CGDL, and Machinations) either to simulate emergent effects within games or to play them autonomously. Only GLiSMo focuses on progression games, however, this
### Table 6.1: Comparison of existing game languages

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Level of Formalism</td>
<td>Formal</td>
<td>Formal</td>
<td>Semi-formal</td>
<td>Formal</td>
<td>Formal</td>
</tr>
<tr>
<td>Type</td>
<td>Emergence</td>
<td>Emergence</td>
<td>Progression</td>
<td>Emergence</td>
<td>Progression</td>
</tr>
<tr>
<td>Representation</td>
<td>Logic-based</td>
<td>Visual</td>
<td>Declarative</td>
<td>Grammar-based</td>
<td>Declarative</td>
</tr>
<tr>
<td>Executable</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Domain</td>
<td>Games</td>
<td>Video Games</td>
<td>Serious Games</td>
<td>Card Games</td>
<td>Gamification</td>
</tr>
<tr>
<td>Target Authors</td>
<td>Theoretical computer scientists</td>
<td>Game designers</td>
<td>Game designers</td>
<td>Theoretical computer scientist</td>
<td>Gamification designers</td>
</tr>
<tr>
<td>Example application</td>
<td>Autonomous play</td>
<td>Simulation of emergence</td>
<td>Documentation of serious games</td>
<td>Generation of novel card games</td>
<td>Standardization, description, and execution of gamification</td>
</tr>
</tbody>
</table>

The language is neither executable nor does the meta-model comply with the requirements for gamification.

Furthermore, no analyzed language has included elements to model the visual aspects of games or gamification. This comprises, for example, the consideration of visual elements (e.g., describing texts or images) and interaction patterns of game mechanics (e.g., rule-driven, user-driven missions). Hence, this thesis not only formalizes the concepts of gamification in a rigorous manner but also presents a language that describes the patterns of user interaction which is not provided by the existing approaches.

### 6.2 Gamification Solutions

In conjunction with the gamification process and the design goals, valid instances of GaML are intended to be automatically translated into code that can be executed by gamification runtime environments (DO6). This section, therefore, analyzes existing gamification runtime environments with regards to the requirements.

In this analysis, 29 gamification systems or runtimes are analyzed and compared against the functional requirements of GaML (Section 5.2) and non-functional requirements as defined for enterprise information systems in general (Section 4.3). Based on the non-functional attributes, existing systems can be classified into three main categories, namely Integrated Gamification Solutions, Achievement Systems, and Generic...
Gamification Platforms. For each of these classes, the following text describes its general functionality, presents implications, and gives examples.

### 6.2.1 Achievement Systems

Achievement Systems (ASs) are defined as “secondary reward systems that have been developed for digital games. They integrate functionality that adds sub-goals to the actual game experience” [141]. In conjunction with the definition of [106], these achievement systems usually add an additional progression dimension on top of the real game. A first academic implementation of an achievement system was presented in [141]. However, the authors focus mainly on the evaluation of achievements rather than the description of the framework. Therefore, [90] extended the concept by rigorously defining the aspects of achievements and conceptualizing a framework. However, a technical solution was not proposed.

Schematically, the AS is external to the actual system and exposes the domain entities via standardized interfaces to the IS (Figure 6.3). Domain entities such as points or badges are defined at design-time. At runtime, when the IS decides to update the player’s state (e.g., by giving a reward), it requests the AS to store this data. Vice versa, the IS may retrieve the player’s current state from the AS, e.g., to display the data in the frontend of the application. Moreover, the achievement system may offer another separate client that visualizes the player’s gamification data in profiles or leaderboards for each game individually.

The major drawback of ASs is that rules are not supported and the gamification logic has to reside within the IS. Consequently, a part of the gamification data, e.g., interim gamification results, resides within the IS as well. This limits their flexibility with regards to design changes as well as reusability in arbitrary contexts. Moreover, the IS has to be modified intensively to integrate with the achievement system which contradicts with the invasivity requirement. In all analyzed cases, integration has to be done programmatically without additional support (medium integrability). Since only parts of the gamification data are available to the AS, analyzability is only given to some extent. None of the analyzed systems supports manageability of data.

From 29 evaluated systems, 11 were identified as ASs (Table 6.2). Implementations of ASs for games are: XBOX LIVE, PLAYSTATION NETWORK, STEAM, IOS GAMECENTER, and BEINTOO. Explicitly designed for gamification are the systems GAMINSIDE, BIDOOR.COM, MPLIFYR, LEADEROARDED, USERINFUSER, and OPENBADGES. As shown in Table 6.2, none of the existing systems allow the definition of all concepts which are necessary for gamification. Almost all systems, however, allow the definition of achievements and points.
In all other aspects, the supported features are strongly depending on the domain where the system is typically applied (e.g., games, loyalty, and reputation manage-

---

1 Legend: ✔ Supported; ✔ Supported, but not changeable; - Not supported; ○ No information available
6.2 Gamification Solutions

For example, systems that are specialized for loyalty programs typically feature redeemable goods and some kind of virtual currency which is not the case for all other ASs.

Furthermore, it is interesting to observe that the majority of the analyzed ASs (e.g., XBOX LIVE, BEINTOO, MPLIFYR, BIGDOOR.COM, or LEADERBOARDED) even come with a predefined value system which cannot be customized, i.e., currencies or point categories are statically defined in order to establish an accountable virtual currency across several games connected to the same AS. In contrast, USERINFUSER or BIGDOOR.COM are highly customizable ASs. Some systems provide a limited number of gamification concepts only, such as points and leaderboard (e.g., LEADERBOARDED) or badges (e.g., OPENBADGES).

6.2.2 Integrated Gamification Solutions

An Integrated Gamification Solution (IGS) has been developed directly for and within the IS (Figure 6.4). Hence, the solution is tied to the application's structure, interfaces, and semantics. Therefore, implementation is very fast at early stages of development since features required by the target application have to be implemented only.

![Figure 6.4: High-level architecture of an integrated solution](image)

It is argued that the overall performance of this approach is high, since all calculations can be directly processed within the system. Similarly, analyzability can be considered as high, since all data is available for analysis within the host application. Due to the tight integration, integratability is obsolete to be considered.

However, it is argued that this approach does hardly scale horizontally. For instance, reusability is not given and, thus, the gamification solution has to be implemented repeatedly for each system. In all surveyed cases, these systems are also domain-specific, i.e., they ship with a set of parameterizable game design elements and rules tied to a particular domain such as sales or sustainability. Therefore, their flexibility is limited to a large extent. Due to the tight coupling of ISs and gamification, invasivity is high. In addition, in no case manageability of gamification data was observed. Finally, the approach results in data silos, i.e., users can hardly share their gamification data across application boundaries without additional efforts since no generalization, standardization, or interoperability is considered. Table 6.3 lists current examples of IGSs for various domains.

Overall, seven systems are classified as IGS, namely GIGYA, PLAYVOX, PRACTICALLY GREEN, ZURMO, CELLCAST, RESULTS.COM, and PUNCHTAB. Due to their tight integra-
tion with the host application, each system is also very specific with regards to the game mechanics it features, i.e., the degree of customization is comparably low. For example, PLAYVOX focuses on specific game mechanics for call centers, GIGYA on social communities, ZURMO on CRM applications, or CELLCAST on learning management solutions.

In contrast to ASs, some of the IGSs allow the definition of very simple rules to customize the gamification concept. For example, GIGYA’s system allows customizing a user action, specify a point consequence, and also some specific mechanics to prevent exploitation (Figure 6.5). Furthermore, actions are conjunctively connected to form a mission, i.e., the mission is done once all actions have been finished. However, these features only support a limited amount of the requirements as identified in this thesis. For instance, it is not possible to filter user actions further based on arbitrary attributes, define relationships between user actions, define consequences upon mission completion, or define arbitrary mission paths.

![Figure 6.5: Action definition example from the Gigya solution [85]](image)

Other approaches, e.g., ZURMO, PRACTICALLY GREEN, or PLAYVOX come with a predefined gamification concept including predefined goals or point values for predefined actions (e.g., [203]), i.e., these gamification concepts can hardly be customized to the specific needs of the organization where it is deployed. However, contradicts with the flexibility and reusability requirements of this thesis. Table 6.3 presents the features of all analyzed IGSs with regards to the functional requirements.

It is shown in Table 6.3 that all analyzed systems support at least to some degree the management of points, achievements, levels, and context in general. However, all analyzed systems are domain specific and, thus, support only a minority of the required features as defined in Chapter 5. As another consequence, none of the existing systems is able to introduce gamification in a highly customizable and reusable manner.
### 6.2 Gamification Solutions

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E2</td>
<td>(Space)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>(Game Levels)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E4</td>
<td>(Events)</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E5</td>
<td>(Achievements)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E5a</td>
<td>(Point Types)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E6</td>
<td>(Points)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E6a, E6b</td>
<td>(Mission Preconditions, Mission Types)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E7</td>
<td>(Skills)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E8</td>
<td>(Levels)</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E9</td>
<td>(Goods)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E9a</td>
<td>(Good Types)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E10</td>
<td>(Roles)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E11</td>
<td>(Leaderboards)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E12a</td>
<td>(Boolean Conditions)</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(✓)</td>
<td>✓</td>
</tr>
<tr>
<td>E12b</td>
<td>(Badge Conditions)</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E12c</td>
<td>(Level Conditions)</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E12d-E12k</td>
<td>(Item - Role Conditions)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E12l</td>
<td>(Event Conditions)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E12m-E12r</td>
<td>(Numeric Conditions)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E13a</td>
<td>(Events)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E13b</td>
<td>(Points)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E13c</td>
<td>(Notifications)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E13d-E13h</td>
<td>(Narrations - Skills)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>CR9</td>
<td>(Narrations)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>CR13</td>
<td>(Context)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
6.2.3 Generic Gamification Platforms

A Generic Gamification Platform (GGP) is completely decoupled from the IS, i.e., the gamification’s state and logic are kept as a separated aspect (Figure 6.6). This allows for an almost non-invasive introduction of gamification into the IS. In all cases, generic platforms offer designers a large set of L1 entities. In GGPs, designers configure the gamification in the platform, i.e., defining rules and metadata prior to runtime in a declarative manner. In all analyzed cases, this process is supported by tools and provides a high degree of flexibility along the gamification process.

During runtime, the platform collects arbitrary user actions from the application and uses the rules to reason over the users’ events considering previous context data of the users.

From a non-functional perspective, the platform can be reused across multiple systems within the company or even across companies. This allows for the definition of standardized interfaces and design artifacts and makes interoperability between information systems easier. Moreover, integrability is highly supported through gamification components and widgets offered by all analyzed GGP implementations.
However, such a generic service approach is also the most challenging one from a conceptual and technical perspective since many different design methodologies have to be supported. Additionally, due to the complete separation of IS and gamification, ensuring high performance is complicated. For example, user actions might be lost or delayed while transmitted between IS and GP.

The analysis classified three systems into this category, namely BUNCHBALL, BADGEVILLE, and IACTONABLE (Table 6.4). As all platforms are considered mature and support different design methodologies, each platform is explained in detail below.

### 6.2.3.1 BUNCHBALL

Bunchball natively supports a variety of L1 game mechanics such as various point categories, badges, user actions, virtual items, real goods etc. However, roles, skills, and narration are not supported. The overall gamification concept is structured by so-called challenges (missions) that determine a linear progression path divided into several game levels. Other mission structures are not supported.

Furthermore, each challenge comprises a set of conditions determining the completion status of the mission. Here, either all conditions in sequence or without sequence, or any of the conditions have to hold in order to complete the mission. According to [42], there are five types of conditions supported:

1. The event is of a specific type (e.g., the user has watched a video). This corresponds to GaML element E12l.
2. The value of an event exceeds a particular threshold (e.g., the user has watched a video for five minutes whereas the time amount is the value attribute of the event). This partially realizes GaML element E12m, since no additional constraints on further, possibly multiple event attributes can be defined.
3. The total value from events of the same type exceeds a particular threshold (e.g., the user has watched videos for five hours).
4. The number of events from the same type exceeds a particular threshold (e.g., the user has watched 5 videos).
5. The number of events from the same type exceeds a particular threshold within a defined time period (e.g., the user has watched ten videos in three hours). The latter three types realize partially GaML element E12p, since no additional aggregations functions are available and only the sum function allows to specify a sliding time window.

It is important to note, that BUNCHBALL’s event model only foresees one numeric attribute per tuple. Hence, additional filters (E12m) or aggregations (E12p) are not considered. Furthermore, the event model does not allow for the detection of collaborative or competitive scenarios through joins (E12n) or temporal operators (E12o). For example, it is not possible to detect that the user has to watch five different videos (i.e., videos having different IDs) or more videos than one of his friends to get a particular reward. Furthermore, additional contextual (E12b-E12g), probability-based (E12h), location-based (E12i) or leaderboard-based (E12j) conditions are not supported.

On the positive side, conditions can be arbitrarily correlated by Boolean operators, thus, reflecting E12a. Furthermore, element E6a is partially supported since it is possible to define levels or exactly one other missions as preconditions for the current
mission. The platform also supports the definition of teams at runtime. A team mission in this platform is completed, when any of the team’s players contributed to the respective conditions. Upon completion, the rewards apply to the entire team. Hence, player (E12r) and team belongingness (E12q) are partially supported as well.

Upon mission completion, the user may receive badges (E13e), points (E13b), notifications (E13c), or virtual items (E9a). It is important to note that virtual items in the BUNCHBALL platform are, however, defined differently as in this thesis. For example, such items cannot be used as advantage in the player’s progression but can be collected and exhibited in a virtual room or given away as a gift to others. Other kinds of feedback are not supported.

Finally, the platform itself and its documentation are not publicly available. Hence, it cannot be assessed with regards to technical concepts and its implications. The analyzed gamification features of the BUNCHBALL platform are summarized in Table 6.4.

<table>
<thead>
<tr>
<th>E# / CR#</th>
<th>BUNCHBALL [42, 43, 148]</th>
<th>BADGEBIT [16, 71]</th>
<th>THESIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1 (Space)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E2 (Game Levels)</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E3 (Event)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E4 (Achievement)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E5 (Points)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E5a (Point Types)</td>
<td>✓ (✓)</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E6 (Missions)</td>
<td>✓</td>
<td>(✓)</td>
<td>✓</td>
</tr>
<tr>
<td>E6a (Mission Preconditions)</td>
<td>✓ (✓)</td>
<td>✓ (✓)</td>
<td>✓</td>
</tr>
<tr>
<td>E6b (Mission Types)</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E7 (Skills)</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E8 (Avatar Levels)</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E9 (Goods)</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E9a (Good Type)</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E10 (Roles)</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E11 (Leaderboards)</td>
<td>✓ ✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>E12 (Conditions)</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>E12a (Boolean Conditions)</td>
<td>✓ -</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
### 6.2 Gamification Solutions

<table>
<thead>
<tr>
<th>E# / CR#</th>
<th>BUNCHBALL [42, 43, 148]</th>
<th>BADGEVILLE [16, 71]</th>
<th>THESIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E12b</td>
<td>(Badge Conditions)</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E12c</td>
<td>(Level Conditions)</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>E12d</td>
<td>(Item Conditions)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E12e</td>
<td>(Mission Conditions)</td>
<td>✓ (✓)</td>
<td>✓</td>
</tr>
<tr>
<td>E12f</td>
<td>(Skill Conditions)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E12g</td>
<td>(Point Conditions)</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>E12h</td>
<td>(Luck Conditions)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E12i</td>
<td>(Location Conditions)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E12j</td>
<td>(Leader Conditions)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E12k</td>
<td>(Role Conditions)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E12l</td>
<td>(Event Conditions)</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
</tr>
<tr>
<td>E12m</td>
<td>(Numeric Conditions)</td>
<td>✓ (✓)</td>
<td>-</td>
</tr>
<tr>
<td>E12n</td>
<td>(Joins)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E12o</td>
<td>(Temporal Conditions)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E12p</td>
<td>(Time Windows)</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>E12q</td>
<td>(Team Belong.)</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>E12r</td>
<td>(Player Belong.)</td>
<td>✓ (✓)</td>
<td>-</td>
</tr>
<tr>
<td>E13</td>
<td>(Consequences)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E13a</td>
<td>(Events)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E13b</td>
<td>(Points)</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
</tr>
<tr>
<td>E13c</td>
<td>(Notifications)</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
</tr>
<tr>
<td>E13d</td>
<td>(Narrations)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E13e</td>
<td>(Badges)</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
</tr>
<tr>
<td>E13f</td>
<td>(Items)</td>
<td>✓ -</td>
<td>✓</td>
</tr>
<tr>
<td>E13g</td>
<td>(Missions)</td>
<td>✓ (✓)</td>
<td>✓</td>
</tr>
<tr>
<td>E13h</td>
<td>(Skills)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CR9</td>
<td>(Narratives)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
6.2.3.2 BADGEVILLE

BADGEVILLE qualifies as another GGP as it complies with the non-functional premises. However, the causality of used game mechanics in BADGEVILLE is slightly different to BUNCHBALL. In fact, this platform supports merely a causality chain that comprises three steps. In the first step, users may gain points for predetermined user actions. In the second step, user may earn badges after reaching a particular amount of points. In the third step, users may complete missions after collecting predefined badges. These badges might be received in a particular order or without any order to fulfill the mission in question [71]. Since the platform supports this method only, it is argued that only very few game mechanics and gamification concepts might be implemented with this platform.

In addition to plain user actions, BADGEVILLE allows to define thresholds how often a user action or event can be issued in a given time period (e.g., one day) or a decay between two user events in order to prevent cheating of the users. However, these mechanics are only parameterizable and not customizable to detect arbitrary event-based situations that may occur in real gamification scenarios. Hence, this platform provides a subset of the BUNCHBALL solution analyzed above.

Similar to BUNCHBALL, the technical concept of the platform and its documentation is not publicly available. Hence, this approach is not qualified to be assessed on a technical conceptual level.

6.2.3.3 IACTIONABLE

IACTIONABLE can be considered as a GGP as well. However, while working on this thesis the service was not maintained any further. Although a potential candidate for analysis, this service is excluded from consideration.
6.2.3.4 SYSTEMATIZATION

Based on the descriptions above, Table 6.4 compares the features of the two analyzed platforms against the theoretical requirements. It is shown, that even the most mature platform still possesses technical and conceptual limitations with regards to arbitrary gamification concepts.

This is taken as motivation for the rest of this thesis to describe a novel runtime environment for gamification which is capable of fulfilling the requirements on the one hand and to validate this platform in the context of real-life scenarios on the other hand.

6.2.4 OTHERS

In this category, all other kinds of services for gamification are subsumed. For example, GAMEBOXED and SNOWFLY offer gamble plugins for FACEBOOK or CRM respectively, which can be used by companies for the inclusion on their social network page. When users play these gambles they may earn tokens which can be redeemed for give-away articles.

GAMEIFY offers a central virtual world where companies may offer merchandising activities (comparable to SECOND LIFE). However, in this case neither gamification concepts nor rules can be defined by designers.

Additionally, some providers (e.g., PLAYGEN) offer the implementation of gamification in custom development projects. In most cases, this comprises the implementation of serious games for education or e-learning in certain business-related domains (e.g., security, health). However, these approaches neither try to reduce development or maintenance costs nor do they comply with the non-functional requirements. Nonetheless, such implementations might benefit to a large extent from the solution approaches described in the sections before.

6.2.5 SYSTEMATIZATION

Based on the individual characterization above, all functional and non-functional attributes between the different approaches are aggregated in Table 6.5. With regards to functional requirements this overview provides a general tendency only, since features are implementation-specific as presented above.

According to Table 6.5, all solutions support at least partially the definition and management of entities, users, and its states. However, all analyzed solutions define their own data model and interfaces on L1. Thus, it is argued that the proposal of GaML provides a first step towards standardization of the identified gamification concepts. As GaML allows to quantitatively assess existing gamification solutions with regards to the requirements defined by the envisioned gamification concepts, it is argued that IT experts benefit from GaML in the provisioning phase of each project as they can objectively decide for the correct solution.

Only GGPs and some IGSs allow the management of gamification rules. However, the powerfulness and expressiveness of rule reasoning is limited in current embodiments as shown in the previous sections. Hence, the technical realization of arbitrary and flexible gamification concept is limited to a large extent.

With regards to non-functional requirements, GGPs, moreover, provide the highest degree of flexibility, non-invasivity, reusability, integratability, analyzability, and manageability. On the other side, technical complexity and performance might be considered critical.
Contrarily, IGSs provide high performance and low complexity but can be considered critical in all other non-functional dimensions.

A trade-off between these distinct approaches are ASs which offer a medium degree of non-functional attributes. However, they do not provide the definition of gamification rules at all. Therefore, they might be only applicable if most of the gamification logic can be implemented in the information system.

Quality attributes of approaches from the other systems are not listed as they are not directly comparable against the requirements.

---

2Legend: ✓ supported; (✓) partially supported; - not supported, ↑ high; → medium; ↓ low; ◦ not considered
7 GAMIFICATION RUNTIME ENVIRONMENT CONCEPT

In this chapter, the thesis’ approach for a gamification runtime environment that complies with the requirements introduced in Chapter 4 is presented. This chapter consists of three main parts. First, the conceptual architecture is presented and trade-offs are discussed. Second, the data model for the technical representation of the required domain entities is shown. Third, based on the architecture and the data model, the translation of GaML into executable code of the runtime environment is presented.

7.1 GENERAL SCENARIO

For this chapter, the architecture shown in Figure 7.1 is assumed as general scenario. This scheme is based on the generic gamification platform assumption introduced in Section 6.2.3.

The proposed gamification platform consists of two main parts. First, an Event Processing Agent (EPA) that holds a set of gamification rules which reason over incoming events to calculate subsequent states. Second, a so-called BEP that manages and maintains the state of domain entities.

Figure 7.1: General scenario for the technical introduction of gamification into enterprise information systems
At design-time, designers or psychologists define the gamification logic in the EPA using some declarative rule language. Domain entities such as badges, points, or levels have to be defined using the BEP. Furthermore, the enterprise systems have to be configured or instrumented to send events to the gamification runtime environment. For example, the BPM engine which manages the enterprise’s services might be configured to send all relevant events to the gamification runtime environment.

At runtime, users (e.g., employees) interact with the enterprise systems. Based on these interactions, the enterprise systems send events to the EPA of the gamification platform. The gamification rules deployed in the EPA calculate feedback such as point or badge instances for the respective users and update the entities’ states using the BEP. The enterprise systems may retrieve the users’ current state, e.g., current points or leaderboards, from the BEP.

This general scheme is used as basis for the discussion below.

### 7.2 SOLUTION APPROACHES

Based on the foundations of Section 2.2, this section discusses the characteristics of two dichotomous data management technologies as solution approaches, namely relational databases and complex event processing, and reflects them in the context of the stated requirements. In general, both approaches offer methods and tools to realize the gamification system and, thus, are candidates for the final architecture. However, it is also shown that none of the approaches on its own suffices all requirements as favored.

#### 7.2.1 RELATIONAL DATABASES

The first solution approach concerns traditional relational databases. Due to ACID (atomicity, consistency, isolation, durability) properties these systems comply with persistence and consistency requirements [25]. In general, databases allow for the definition of transient queries on persistent data. This permits the definition of ad-hoc queries and ex post analysis and, therefore, complies with the stated analyzability requirements. Furthermore, well-established mechanisms and technologies for application-level entity management, e.g., object relational mapping, exist (manageability).

However, databases typically fall short when large amounts of continuous data such as events have to be analyzed. Without index the correlation or joining of many events becomes slow for larger data sets. While the introduction of indexes as in-memory structures may provide significant speed up, the update of indexes can be costly and inefficient on high event rates. In addition, events leading to an index update might be never used again, i.e., upon arrival of an event, the index is updated at significant cost but the event might be never looked-up again. Thus, the cost-benefit ratio of the created indexes might be large.

Moreover, the efficient definition of indexes is not possible since the nature of events (e.g., interarrival times, probabilities, or payload) cannot be assumed beforehand since this strongly depends on the application, use cases, and user behavior. Thus, indexes might be created reactively, for example, based on historical query statistics. Hence, real-time analysis requirements are insufficiently supported.

---

1\(O(\prod_{i=0}^{n} r_i)\) or at least \(O(\sum_{i=0}^{n} r_i \cdot \log(r_i))\) for merge joins with \(r_i\) being the number of pages for the \(i\)th relation attribute [63].

2For instance, \(O(\sum_{i=0}^{n} r_i)\) assuming at least one equijoin [63].
Finally, different types of events may have different signatures. In databases, the various event types need to be represented by using separate tables. Altering the signature of events or introducing new event types requires the restructuring and migration of the database schema. Since this is a cumbersome and error-prone activity, flexibility with regard to design changes is strongly limited.

### 7.2.2 COMPLEX EVENT PROCESSING

In contrast to databases, complex event processing technologies require the definition of persistent queries which are issued on transient and continuous event or data streams, i.e., data tuples are volatile in-memory structures available for analysis for a particular time period only. In production rule systems or Event-Condition-Action (ECA) rule systems, fast pattern matching algorithms are available (e.g., RETE [81]). Furthermore, ESP systems introduce non-blocking aggregations through finite windows. Due to their reactive nature, zero latency between the arrival of a tuple and processing can be assumed, i.e., events are analyzed immediately rather than being stored first. Since all aforementioned algorithms and the data reside in the main memory, the computation of results is very fast. Thus, these technologies are considered to realize and support at least soft real-time requirements [15].

Furthermore, the definition of queries in the form of production or ECA rules allows for a high degree of flexibility due to their dynamic structure, i.e., a static data or query schema is not required. For example, in RETE-based systems the internal propagation graph representation (Section 2.2.4) can be modified on the fly, e.g., by adding or removing additional nodes or reusing existing ones. In ESP systems, operators in a query plan can be exchanged at runtime as well. If newly introduced operators with empty window need tuples from other operators (e.g., from an exchanged operator), efficient migration algorithms are available to transfer tuples between operators at runtime avoiding downtime for migration (e.g., [198]).

However, in CEP technologies the data has to reside within the working memory. Besides the fact that data in main memory is volatile, all modern CEP technologies have to fulfill either implicit (e.g., using temporal constraints, finite windows) and explicit memory management (e.g., through declaration of event expiration durations), or load shedding (e.g., [15]) on the unprocessed data to reduce memory footprint. For most practical scenarios, this makes ex post analytics and ad-hoc queries impossible and, therefore, contradicts with the stated analyzability and data manageability requirements (Section 4.3).

Finally, implementing entity management logic directly within rules bloats an application's rule memory and makes it difficult to read and maintain. This strongly suggests for a decoupling and encapsulation of logic in entities [144].

### 7.3 HYBRID APPROACH

The last sections have shown that the characteristics of the considered approaches do not comply fully with the non-functional requirements stated in Chapter 4. Consequently, this section proposes a hybrid approach which aims at system that unifies the advantages and compensates the disadvantages of the considered technologies. The derived system is used as a general architecture pattern for the implementation of the gamification runtime environment later.
7.3.1 GENERAL CASE

The general idea of a hybrid system is shown in Figure 7.2. This system comprises several event sources, event sinks, one EPA, as well as one BEP.

The BEP is a container managing so-called business entities which can be considered as typed data structures containing state and relationships to other entities. These entities can be mapped and stored into relations of an arbitrary underlying relational database. For example, the BEP manages user entities holding status information for all users of the gamification system \( \{U_1, ..., U_k\} \) (Table 7.1).

On the one hand, the BEP offers an update interface that allows for updates to the state of the domain entities. In general, the interface's syntax and semantics depends on the intended business functionality in order to reflect the functional requirements of the solution. For instance, in the gamification domain the BEP may offer interfaces to receive low-level updates for points or badges through the generic update interface.

On the other hand, a query interface might be utilized to retrieve the entities’ current state (e.g., the user’s average points). Finally, the BEP may provide derived state in transient data structures upon queries, e.g., an individual and temporary high score for single users or several leaderboards across multiple users (Table 7.1).

Within the EPA, multiple gamification rules are deployed. In this thesis, rules and corresponding examples are based on production rules with integrated event processing capabilities, i.e., each rule comprises a LHS and RHS representing a rule's condition or consequence respectively (Section 2.2.4).

Table 7.2 presents examples of typical LHSs of gamification rules in the hybrid approach. These examples include the presence of an event only \( \circ \), Boolean \( \circ \) and temporal \( \circ \) operators with events as operands, or event aggregation \( \circ \).

Furthermore, the example comprises LHSs which require additional non-event data, i.e., state data from the domain entities. This contextual data might be processed...
### Example

<table>
<thead>
<tr>
<th>Example</th>
<th>Formal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>$U_1$</td>
</tr>
<tr>
<td>Set of users</td>
<td>${U_1, ..., U_k}$</td>
</tr>
<tr>
<td>Set of points for User $U_i$</td>
<td>${p_{i1}, ..., p_{in}}$</td>
</tr>
<tr>
<td>Set of badges for User $U_i$</td>
<td>${b_{i1}, ..., b_{im}}$</td>
</tr>
<tr>
<td>Sum of points for User $U_i$</td>
<td>$P_{U_i} = \sum_{j=0}^{n} p_{ij}$</td>
</tr>
<tr>
<td>Average of points for User $U_i$</td>
<td>$\bar{P}<em>{U_i} = \frac{1}{n} \sum</em>{j=0}^{n} p_{ij}$</td>
</tr>
<tr>
<td>Leaderboard between users</td>
<td>$(U_1, ..., U_k)$</td>
</tr>
</tbody>
</table>

Table 7.1: Examples for gamification-related state information managed by the business entity provider

upon event arrival such as in ECA rules $\circledast$ (e.g., an event $e_i$ occurs in conjunction with the user’s average points $\bar{P}_{U_i}$ being equal or greater than 20).

Ultimately, such contextual data might be processed without explicit event occurrence such as in production rules $\circledcirc$ (e.g., if a user $U_i$ holds any of the badges $b_{i1}$ or $b_{i2}$).

<table>
<thead>
<tr>
<th>LHS Types</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\circledast$ Simple Event / Event Rule</td>
<td>$e_1 \rightarrow ...$</td>
</tr>
<tr>
<td>$\circledast$ Boolean event correlation</td>
<td>$e_1 \land e_2 \rightarrow ...$</td>
</tr>
<tr>
<td>$\circledast$ Temporal event operators</td>
<td>$e_1$ during $e_2$ $\rightarrow ...$</td>
</tr>
<tr>
<td>$\circledast$ Event Aggregation</td>
<td>$\frac{1}{n} \sum_{i=0}^{n} e_{i}^{\text{value}} &gt; 20 \land e_2 \rightarrow ..., n = \text{size of window}$</td>
</tr>
<tr>
<td>$\circledast$ Event with Context</td>
<td>$e_1 \land P_{t,U_i} \geq 20 \rightarrow ...$</td>
</tr>
<tr>
<td>$\circledast$ Context only</td>
<td>$U_i \land (b_{i1} \lor b_{i2}) \rightarrow ...$</td>
</tr>
</tbody>
</table>

Table 7.2: Examples for gamification-related conditions managed by the event processing agent

Table 7.3 shows general examples for the RHS of rules. Here, either complex events $\circledcirc$, domain data $\circledast$, or combinations thereof $\circledast \circledast$ might be generated.

In the first case $\circledast$, these events may represent newly created or derived events. As explained in Section 2.2.3.2, derived events may contain some of the data of its activating events. For instance, a derived event might be an event that is enriched by a temporal duration.

In the second case $\circledast$, domain data can be considered as a special type of event as they are assumed to be not required in any of the rule’s LHS directly but might be processed implicitly after they have updated the state of the issuing entity.

This ends the discussion on the structural parts of this hybrid approach. In the following text, the interaction procedure between these parts is elaborated in more detail.
The sequence of steps is highlighted in Figure 7.2. Initially, the EPA may receive events from arbitrary sources in step $\oplus$. Assuming that the BEP does not contain any data at the beginning, only rules of types $\ominus$ to $\oplus$ can be activated. Activated rules may cause the generation of new events $\ominus$, domain data $\ominus$, or combinations thereof $\ominus$ in step $\ominus$. While the creation of new (possibly complex) events can be processed directly, domain data needs to be synchronized with the entities in the BEP. The rules’ RHS, therefore, may call the update interface of the BEP in step $\ominus$. After updates have been received, validated, and stored by the BEP in step $\oplus$, data can be forwarded to various event sinks deployed in the enterprise in step $\odot$, e.g., to notify the user on the successful completion of a task. Some rules in the EPA, i.e., rules of type $\ominus$ and $\ominus$ may need contextual state data from the BEP. Hence, the event processor may retrieve and evaluate state by utilizing the BEP’s query interface in step $\ominus$. This, in turn, may lead to the activation of new rules which closes the processing cycle of rules. Finally, all processed events with point-in-time semantics (Section 2.2.3.2) are retracted from the EPA’s working memory.

By contract, all point-in-time events are removed from the EPA after each evaluation cycle for two reasons. First, in gamification scenarios most events have point-in-time semantics, thus, their retraction reduces memory consumption in the long run. Second, if such events are not automatically managed by the EPA, event management is implicitly shifted to the responsibility of the rule author, i.e., the event management is mixed with the business logic and, thus, not transparent to the rule author anymore. This, however, possesses limitations with regards to rule design.

For example, consider at least two rules which are activated by the same event, i.e., both rules react on the same event in their LHS. If event management is not transparent, the rule author has to decide when the event is retracted from the internal working memory. Retracting the event in one of the rule’s RHS, however, is not possible without harming the business logic. If the event is retracted by the first rule’s RHS, the activation of the second rule will be canceled and the corresponding RHS is never executed. This applies also for the opposite case. Hence, the rule author has to retract the event manually after all rules are executed as expected. This introduces additional overhead for writing and testing such memory management rules and is an error-prone activity. Therefore, event management is done automatically based on the semantics of events.

However, interval events are not automatically retracted after each evaluation cycle as they imply a particular time period in which they are valid. Depending on the configuration such events are either managed by the rule author or automatically, by utilizing a generic rule which retracts each interval event after its duration has expired.
Table 7.4 summarizes the mapping of the various rule types onto the corresponding parts of the hybrid architecture. All state data encapsulated in domain entities (Table 7.1) is managed by the BEP. Within the EPA, rule types \( \odot \) to \( \odot \) can be directly reflected. The update interface between BEP and EPA allows for the additional rule types \( \odot \) and \( \odot \) within the EPA. Finally, rule types \( \odot \) and \( \odot \) are possible in the EPA when a query interface between both components is present.

While the presented hybrid system might be effective in general, the efficiency of the approach, however, mainly depends on the communication paradigm that is used to communicate between EPA and BEP. Therefore, different communication solutions and their advantages and disadvantages are described in the next sections.

### 7.3.2 SYNCHRONOUS CONTEXT-UPDATE

Figure 7.3 shows an adapted version of the general hybrid approach introduced in Figure 7.2. In particular, the synchronous communication algorithm is presented in four sequential steps.

Events arrive at the EPA in step 1. Only rules of type \( \odot \) to \( \odot \) can be activated in step 2. If at least one rule of type \( \odot \) or \( \odot \) is activated, the update interface of the BEP is called synchronously in step 3, i.e., the EPA halts until the BEP acknowledges the message. Since it is assumed that the utilization of the BEP’s update interface yielded a change in one of the domain entities, another evaluation cycle is triggered at the EPA. This leads to the potential activation of rule types \( \odot \) and \( \odot \) as the rule engine utilizes the BEP’s query interface while evaluating the rules’ LHS in step 4.

The major advantage of this solution is that due to the synchronous nature, both communication partners are always consistent with each other, as long there is no network partition in between. Hence, multiple requesters may use the BEP’s update and query interfaces concurrently. For example, it is possible to change the user’s number of points in the BEP via an external interface concurrently to the reception of additional events of this user in EPA. Moreover, it is also possible to enable multiple EPAs, for example, to distribute event processing on multiple instances working in conjunction with the same BEP.

However, two main disadvantages arise out of this design. First, due to blocking behavior of synchronous calls, the rule execution and update operations are delayed. Hence, the response time might not be sufficient even for moderate workloads. Furthermore, when sampling rates of the event sources are high, i.e., there are many events to be processed concurrently, the request queues of the update and query interfaces at the BEP may overload and start to reject requests after a particular timeout period. This leads to the loss of tuples and wrong results with regards to the business
logic. Second, situations can occur where events may cause the querying of the BEP, although no update took place. This results in many unnecessary calls to the query interface and, thus, introduce further delays.

Although the synchronous approach works in general, the solution might be very inefficient, especially when large amounts of events have to be processed and many rules of type $\delta$, $\epsilon$, $\eta$, and $\iota$ reside within the rule engine. Thus, the approach heavily contradicts the real-time requirements (Section 4.3).

### 7.3.3 ASYNCHRONOUS CONTEXT-UPDATE

Alternatively, the communication can be conducted asynchronously. While this may speed up communication in general, race conditions leading to wrong results and inconsistencies may occur when domain data from the BEP is required.

The first example scenario concerns one single process (e.g., the EPA) as shown in Figure 7.3 which issues an update (step 3) followed by a query (step 4). When at least the update is done asynchronously, there is no guarantee that the subsequent query returns the correct result since the query might return a result without the update having taken place. Such inconsistencies may lead to further errors. For example, if the query returns an inconsistent result, an unintended consequence (e.g., another update) might happen.

To avoid this scenario, a proxy component (called BEP proxy) is introduced in the EPA that manages the data between EPA and BEP more efficiently and acts as central synchronization point (Figure 7.4). This proxy implements the same interface specification of the update and query interfaces defined by the BEP. Hence, this proxy takes care of the same entity management functionality as the BEP except persistence of the occurring data.

Compared to the synchronous case, the EPA only communicates with the local BEP proxy. All updates and queries are issued against the proxy. In the update step 3, the proxy stores the data internally and lazily replicates the data to the BEP in step 4. Vice
versa, query results might be retrieved directly from the proxy. After starting the EPA (e.g., after a failure or a restart) the BEP proxy can be always reconstructed from the persistent data of the BEP in step 5.

This approach avoids race conditions between exactly one EPA and BEP. Furthermore, it heavily accelerates the communication by several orders of magnitude (see Chapter 8) compared to the synchronous case. However, the low-level updates in the proxy force a constantly growing memory. Hence, there is a space-time trade-off which is characterized in Section 8.4.2. Additionally, the second distributed scenario as described below is not sufficiently covered.

In the second scenario, multiple processes may modify data in the BEP concurrently. These changes can be triggered by multiple different EPAs or by an external authority directly within the BEP (e.g., an administrator). In such cases, there is no guarantee for one process to contain the concurrent updates from all other processes. Since a global synchronization point is missing, the individual processes may operate on stale data.

As an idea, different locking strategies (e.g., pessimistic or optimistic locking) have to be applied to synchronize the access of multiple processes accordingly. However, traditional transactional scopes have not been investigates for reactive production rule or event-stream systems so far. Thus, these problems need to be addressed by future research and are beyond the scope of this thesis.

7.4 BUSINESS ENTITY PROVIDER

Until now, the BEP has been considered domain-independently. However, for the rest of this thesis, it is necessary to describe gamification-specific adaptations. This includes the concrete description of the underlying data model and the query and update interfaces as postulated by the hybrid solution.
7.4.1 DATA MODEL

Every BEP is characterized through a data model representing the conceptual entities on a technical level. Figure 7.5 presents the entity model as class diagram in Unified Modeling Language (UML). As such, the data model mainly reflects the concepts of GaML and their relationships as introduced in Chapter 5 except gamification rules. In addition to the entities, association classes reflect concrete instances of a concept. For example, the class Badge2Avatar represents properties such as the date when the badge was obtained, an optional reason, and if the player exhibits the badge in a so-called showcase element. These information are stored for each badge that one player receives at runtime. Note that in Figure 7.5 standard UML semantics apply, i.e., associations with no explicit cardinalities refer to many-to-many relationships. Furthermore, this data model refers to some specific data types such as AGGREGATION or POINTTYPE which are defined as enumerations presented in Figure 7.6.

Additionally, some specific functional runtime requirements are reflected in this data model. In particular this concerns conceptual requirements CR9 (Narration), CR13 (Context), CR15 (Avatar), CR16 (Marketplace), and CR19 (Team) which have been defined as requirements for the runtime system (Chapter 4). First, the avatar concept CR15 is introduced reflecting several virtual representations that a player can have. For this class, the user may decide if the avatar is publicly shared or which avatar is used by default. Furthermore, the majority of classes is associated with the avatar class such as point, badge, or role. Second, the data model reflects the context CR13 of all entities based on the various association classes. Third, the team class CR19 is introduced. For the sake of brevity, however, it is only associated with the point class in Figure 7.5. Nonetheless, it is important to mention that all classes which have an association with the avatar class are also associated with the team class. Fourth, the marketplace CR16 concept is introduced. In this entity, users may offer their earned virtual items or real goods to others, either for free (e.g., as a gift) or for a self-defined price. The according class is used to track all ongoing transactions that may take place over time. Finally, narrations are recorded for each avatar instance. Similar to the notifications, narrations contain attributes for the text, title as well as a link to an optional image. Moreover, Boolean flags signify, whether the messages have been read by the user or not.

In addition to these functional requirements, the data model reflects further non-functional requirements as determined in Section 4.3. For example, there are three many-to-many associations between the avatar and the point entity. While the first reflects the assignment of new points as required in standard gamification scenarios, the others implement important runtime requirements. The AggregatedPlayerPoints class is used at runtime to store the user’s aggregated points, i.e., points are aggregated from the Points2Avatar class for all different aggregation methods (e.g., sum, average, min, or max) and deleted afterwards. Consequently, the user’s overall points have to be calculated from both relations. In addition, the Redemption class stores all points that were spent by the user at runtime (e.g., for buying or trading real or virtual goods).

Finally, it is important to note that Figure 7.5 reflects only a part of the entire data model. In fact, the figure presents an avatar-centric perspective, i.e., the root element is the Avatar concept in relation to all other concepts. For the sake of readability, some entities have been omitted. A similar diagram could be drawn with the Team entity as the root entity. This is, for example, indicated by the presented team entity which references the Point entity in a many-to-many association which has a class TeamPoints attached. The same applies for all other entities which relate to the Team entity as defined by GaML. Furthermore, Figure D.1 shows the associations between entities and the image entity which has been omitted from Figure 7.5 as well.
Additional semantics in this data model are not explicitly presented as they completely comply with the static semantics (e.g., unique constrains, not null constraints, value ranges, referential integrity of associations) given in Chapter 5. However, additional processing functionality and derived (transient) data structures and queries might be defined on top of this data model. These aspects are considered in the query and update interfaces of the business entity provider as described in the following sections.
7.4.2 UPDATE AND QUERY INTERFACES

On the one hand, the update interface comprises four kinds of methods. First, methods to create or insert new domain concepts as meta-data (e.g., badges or points). As shown in the example below, these methods require a certain classID, which uniquely identifies the respective domain entity, and a list of key-value pairs for the corresponding properties of that domain entity:

\[
\text{create(ClassID, } \text{property1}=\text{value1}, \ldots, \text{propertyN}=\text{valueN}) : \text{INT}
\]

Second, methods to insert concrete instances of the concepts, i.e., tuples which are inserted into one of the association classes. Besides simple insertion, these methods have to implement the specific semantics determined by the various game mechanics. For example, a point update for a player’s avatar has to be checked with regards to the point semantics, i.e., the point amount in the update message must be positive in case it refers to an advancing point type.

\[
\text{givePoints(ID, amount=\text{val}, avatar=\text{val}, reason=\text{val}) : BOOLEAN}
\]
\[
\text{giveBadge(ID, avatar=\text{val}) : BOOLEAN}
\]
\[
\text{giveMission(ID, avatar=\text{val}) : BOOLEAN}
\]

Third, methods to delete or update domain concepts and cascading the operation to all its corresponding children when required.

\[
\text{delete(ClassID, ID) : BOOLEAN}
\]
\[
\text{update(ClassID, ID, prop1=\text{val1}, \ldots, propN=\text{valN}) : BOOLEAN}
\]

Fourth, miscellaneous methods for administration purposes. As those are very implementation-specific they are considered later in the evaluation (Chapter 8).

On the other hand, the query interface offers methods to retrieve the state of the domain entities or transient data structures derived from the entities. First, so-called getter methods return all instances of an entity, i.e., an avatar or team:

\[
\text{getPointsForAvatar(pointID=\text{val}, avatarID=\text{val}, agg=\text{val}) : Score}
\]
\[
\text{getBadgesForAvatar(avatarID=\text{val}) : List}
\]
\[
\text{getMissionsForAvatar(avatarID=\text{val}) : List}
\]
\[
\text{getAvatarLeaderboard(pointID=\text{val}, order=\text{val}) : List}
\]
\[
\ldots
\]
\[
\text{getPointsForTeam(pointID=\text{val}, teamID=\text{val}, agg=\text{val}) : Score}
\]
\[
\text{getBadgesForTeam(teamID=\text{val}) : List}
\]
\[
\text{getTeamLeaderboard(pointID=\text{val}, order=\text{val}) : List}
\]
\[
\text{getMissionsForTeam(teamID=\text{val}) : List}
\]

For example, the Score entity returned by the getPointsForAvatar method contains the aggregated amount of points for various aggregation methods and the associated avatar object of the player. As another example, several sorted score objects for multiple avatars are returned by the getAvatarLeaderboard method subsumed in a list structure.

Second, so-called has methods returning true or false if an avatar or team owns at least one instance of the entity in question.

---

3Note that the following examples are specified independently of any programming language. Concrete examples of the update and query interfaces in the Java programming language can be found in Appendix B.
7.5 EVENT PROCESSING AGENT

Within the hybrid architecture presented in Section 7.3, the EPA has been described as a domain-independent component that is capable of rule and complex event processing based on the events received from the enterprise information systems. Similar to the discussion on the BEP, this section introduces additional gamification-specific adaptations for the EPA. Primarily this includes a presentation of the concrete event processing procedure within the EPA as well as the introduction of standardized gamification events and rules which are automatically deployed in every instance of the runtime environment.

7.5.1 EVENT PROCESSING PROCEDURE

The concrete event processing procedure of the proposed gamification system is based on three premises. First, besides events, avatars and teams are considered as first-class citizens in the rule engine as well, i.e., they are not necessarily queried over the BEP’s query interface in each evaluation cycle but remain in the working memory once loaded. This is necessary to avoid additional communication overhead in each evaluation loop. Although it is assumed that these objects change very seldom, the EPA may request the BEP in each cycle to check for possible changes and reload the entities completely, if a change has occurred.

Second, all events are retracted after their validity period (duration), i.e., point-in-time events are retracted immediately after each evaluation cycle and interval events are retracted after their particular validity period has expired\(^4\).

Third, there are reserved event types with predefined semantics in the context of gamification, i.e., they cannot be freely chosen by a designer or rule author as they are used to process standard scenarios (cf. Section 7.5.2).

Figure 7.7 presents the overall processing algorithm which is initiated upon the reception of events from the host application. In a first step, the incoming set of events is converted into two stacks \(LE\) and \(RE\) containing the events in descending order with regards to the events’ timestamps, i.e., the stack’s top element refers to the oldest event, the last event in the stack to the most recent one. Based on the LE, the first event is popped off the stack and basic consistency checks are performed, e.g., if the event contains at least a player ID and that the event type is set.

Subsequently, it is checked if the event is of the special type \(selectAvatar\). This event might be issued by players to initially select their avatar or to change between different avatars. In both cases, the EPA tries to retrieve the avatar from the BEP.

\(^4\)Recall that the validity period of interval events is determined by the event’s occurrence timestamp plus its duration.
If the BEP does not contain the referenced avatar, the player obviously uses it for the first time. Therefore, the avatar is created at the BEP and the instance is inserted into the rule engine. To signify the creation, another special event called newUser is created for the avatar and inserted into the rule engine as well. Standard rules may now react on this newUser event, for example, to assign this avatar the initial missions or rewards (Section 7.5.2).

However, if the BEP contains the avatar, i.e., the user switches to an existing one, this avatar is inserted into the rule engine, if it does not already exist in-memory (e.g., after a system restart). In the last step of both subprocesses, the currently selected avatar is stored in a local lookup table called AvatarMap.

Similarly, the same process is applied for the case that there is no explicit selectAvatar event, i.e., a player who has no stored avatar in the local lookup table issues any other gamification-specific event.

After these cases have been checked, it is ensured that the player’s current avatar is determined. Therefore, the event is enriched by the corresponding avatar ID based on the player’s ID from the original event.

Subsequently, it is checked, whether the event is of type deleteUser which is a special management event to delete an avatar representation from the runtime environment. This deletion affects the BEP using the update interface as well as the local
deletion of the avatar entity from the rule engine’s working memory. In this case, the
processing is already finished and the next element is retrieved from the LE.

Finally, the event is inserted into the rule engine to trigger the rule matching process.

This entire procedure is repeated until LE is empty. Afterwards, the conflict set of
the rule engine is processed (ruleEngine.fireAllRules), i.e., potential updates and
queries as specified by the conflict set are issued against the BEP.

Finally, the RE as copy of LE is processed to check if the events have point-in-time
semantics. If so, these events are removed from the rule engine’s working memory.
The procedure stops when RE is empty.

For the sake of simplicity this algorithm has been described for players and their
associated avatars only. However, all checks in which an avatar ID is involved are also
performed for team entities as well.

### 7.5.2 STANDARDIZED EVENTS AND RULES

In the general case, it has been assumed that the rule author creates all rules from
scratch. However, there are rules which can be considered as essential for every ga-
mification scenario. These standard gamification rules are reported below in Drools
Rule Language (DRL), a well-known language for writing production rules with event
processing capabilities [70].

Listings 7.1 and 7.2 refer to standard rules for the acceptance or rejection of missions
offered to the user. As such, these rules also require the definition of two standard
events, namely acceptMission and rejectMission. When the first occurs, the mission
in question is assigned to the issuing avatar and set to “available.” In the latter case, the
mission in question is assigned to the avatar and set to “rejected.” To cover the same
functionality for teams, similar rules are used (not depicted).

```drl
1 rule 'ACCEPT MISSION'
2 when
3 $a : Avatar($aid : avatarID)
4 $evt : EventObject(type=="acceptMission", data['missionName']!=null, $aid==
     avatarID) from entry-point eventstream
5 then
6 updateAPI.addMissionToAvatar($aid, $evt.get('missionName'));
7 update($a);

Listing 7.1: Core rule for user-accepted missions (accept)
```

```drl
1 rule 'DECLINE MISSION'
2 when
3 $a : Avatar($aid : id)
4 $evt : EventObject(type=="rejectMission", $mname: data['missionName'], $aid==
     avatarID) from entry-point eventstream
5 then
6 updateAPI.rejectMissionForAvatar($aid, $mname);
7 update($a);

Listing 7.2: Core rule for user-accepted missions (decline)
```

Furthermore, Listings 7.3 and 7.4 refer to the automatic retraction of point-in-time or
interval events respectively. In both cases, the rules’ salience values determine the
order of execution when multiple rules have been activated for the same event. Using
the lowest possible integer value for the salience attribute, both rules are executed
with the lowest priority. In the case of interval events, furthermore, the retraction does
not happen immediately, but a timer is started based on the event’s duration. After the
timer expired, the activating event is removed from the working memory.
7 Gamification Runtime Environment Concept

```java
rule 'RETRACT_POINT_EVENTS'
salience Integer.MIN_VALUE
when
  $evt : EventObject(eventDuration==0) from entry-point eventstream
then
  retract($evt);
```

Listing 7.3: Core rule for deleting point-in-time events

```java
rule 'RETRACT_INTERVAL_EVENTS'
salience Integer.MIN_VALUE
timer(expr:$duration;)
when
  $evt : EventObject($duration : eventDuration > 0) from entry-point eventstream
then
  retract($evt);
```

Listing 7.4: Core rule for deleting interval events

Some gamification concepts allow interaction mechanisms, e.g., activating a skill or consuming a virtual item. For these purposes, additional core rules exist. For example, Listing 7.5 shows the core rule which handles the event for activating a skill (activateSkill), given that the skill in question is not active at the moment. If true, the rule’s LHS tries to activate the skill. If successful, the skill is presented as activated on the front-end using the BEP’s update interface. Furthermore, a new interval event activatedSkill is inserted into the rule engine where the duration equals the skill’s availability period which is returned by the activation operation.

According to Listing 7.4, the delayed event is automatically retracted when the availability period of the skill expires, i.e., the player is allowed to activate the skill for the avatar again. Note that in some games, skills may have an additional cool-down period which has to expire before the skill can be reused. This is not reflected in this example, but would be a simple extension to GaML’s meta-model as well as the required rules.

```java
rule 'SRID'
when
  $a : Avatar($aid : avatarID)
  $evt : EventObject(type=='activateSkill', $sname:data['skillName'], $aid==avatarID) from entry-point eventstream
  not(EventObject(type=='activatedSkill', data['skillName']==$sname, $aid==avatarID) from entry-point internalstream)
then
  long duration = updateAPI.activateSkill($aid, $sname);
  EventObject as = new EventObject($evt);
  as.setType('activatedSkill')
  as.setEventDuration(duration);
  entryPoints['internalstream'].insert(as);
  update($a);
```

Listing 7.5: Core rule for activating skills

Similar to the latter case, there are rules for handling the semantics of virtual items. In this case, however, no interval event is created that signifies the activation of the item since items are considered to be consumed after their usage. Nonetheless, for items and goods additional rules are necessary to process redeem and trade events which are used to calculate the redemption of points for goods or the exchange of goods between entities or into points.

Finally, there are core rules that comprise a specific LHS but where the RHS still has to be defined by the gamification or IT expert. For example, Listing 7.6 shows the
rule for newly created avatars. This rule is activated on the standard event newUser (Section 7.5.1). The gamification expert, however, still has to determine what happens when new avatars are created (e.g., the first time they enter the system). By default, no action would take place.

```
1 rule 'newUser'
2 when
3  $a : Avatar($aid: avatarID)
4  $event : EventObject(type=='newUser', $aid==avatarID) from entry-point
   eventstream
5 then
6 ...
```

Listing 7.6: Core rule for new users or avatars

Based on these core rules, Table 7.5 summarizes the corresponding standard event types which have the semantics described above. The table’s second column indicates whether the event is reserved in either case (always) or only if the according mechanisms are configured by the gamification or IT expert (on rule). Moreover, the shown events might be also used in conjunction with other, non-core events for other design purposes. For example, a rule may check, if a player accepted five missions in a row for a particular avatar instance.

<table>
<thead>
<tr>
<th>Type</th>
<th>Reserved</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>newUser</td>
<td>always</td>
<td>reserved to trigger a configurable, initial assignment rule when new avatars are created for the first time.</td>
</tr>
<tr>
<td>deleteUser</td>
<td>always</td>
<td>reserved to trigger a configurable deletion rule, i.e., a rule that determines the consequence when an avatar is deleted by its owner (i.e., user or player).</td>
</tr>
<tr>
<td>selectAvatar</td>
<td>always</td>
<td>reserved to signify that the player selects a different default avatar.</td>
</tr>
<tr>
<td>newTeam</td>
<td>always</td>
<td>reserved to trigger a configurable, initial assignment rule when a new team is created.</td>
</tr>
<tr>
<td>deleteTeam</td>
<td>always</td>
<td>reserved to trigger a configurable deletion rule, i.e., a rule that determines what happens if a team is deleted.</td>
</tr>
<tr>
<td>joinTeam</td>
<td>always</td>
<td>reserved to trigger a configurable rule when a player’s avatar joins a new team.</td>
</tr>
<tr>
<td>leaveTeam</td>
<td>always</td>
<td>reserved to trigger a configurable rule when a player’s avatar leaves a team.</td>
</tr>
<tr>
<td>acceptMission</td>
<td>always</td>
<td>reserved to signify that an entity has accepted an offered mission manually.</td>
</tr>
<tr>
<td>declineMission</td>
<td>always</td>
<td>reserved to signify that an entity has declined to participate in an offered mission.</td>
</tr>
<tr>
<td>activateSkill</td>
<td>on rule</td>
<td>reserved to signify that an entity has activated a particular skill.</td>
</tr>
<tr>
<td>activatedSkill</td>
<td>on rule</td>
<td>reserved to signify that a skill is currently activated and cannot be reactivated.</td>
</tr>
</tbody>
</table>
### 7.6 Compilation Procedure

In the following text, the compilation procedure from valid *GaML* instances into executable code for the generic gamification runtime environment is presented. Derived from the overall structure of this thesis (Figure 1.1), a more detailed procedure is shown in Figure 7.8.

The overall procedure is derived from a typical model driven architecture process which comprises a Platform Independent Model (PIM), Platform Specific Model (PSM), and code level as shown in the lower lane of Figure 7.8 [113]. As the name implies, the PIM is a model which is independent from any specific technology (e.g., gamification system) or computational model (e.g., imperative, abstract state machine, or rule-based) [82, 113]. Here, *GaML* is considered as the PIM.

The PSM is a model which is dictated by a specific technology (e.g., through programming languages, APIs, or tools). In the case of this thesis, the provided interfaces of a specific gamification system or solution are determining the PSM. Using transformers, instances of the PIM can be translated into one or more PSMs [113].

Eventually, the PSM can be translated into running code which can be executed as computer program. In most scenarios, the PSM and code model are closely coupled [113]. For example, a PSM in the Java programming language is inherently coupled with the bytecode compiled from the sources for the Java runtime environment. In this thesis, the code level is represented through a running instance of the proposed gamification runtime system.5

In this section, the translation of *GaML* (i.e., the PIM) into code for the runtime environment (i.e., exactly one PSM) is presented. More precisely, this process starts with a *GaML* instance describing an arbitrary gamification concept (Chapter 5). A parser

---

5 Theoretically, more levels have to be distinguished in practical scenarios. For example, the rule language within the gamification runtime environment enforces again the generation of programming language code on multiple levels. However, for the sake of simplicity the entire system represents the code level herein.
program tries to parse the respective instance. Since the parser is a generated artifact and, therefore, contains no novel concepts, its description is deferred to the evaluation chapter in this thesis (Section 8.2.1).

If parsing is successful, i.e., the instance complies with the syntax, an Abstract Syntax Tree (AST) is created representing the instance. Based on the AST, static semantics as described in Section 5.3 are validated. If these tests pass as well, the GaML instance can be translated into the platform-specific model determined by the gamification runtime environment.

The target model of the gamification runtime system comprises the provided interfaces of the BEP and EPA as presented in Sections 7.4 and 7.5. For the BEP this includes the update, query, and administration interfaces. For the EPA this includes the event and administration interfaces for creating rules. Furthermore, the rule engine which is included within the EPA defines its own production rule model. Therefore, the translation of GaML into runtime code comprises two steps.

First, GaML rules, i.e., conditions and consequences, have to be translated into a specific rule language for the EPA’s rule engine. Throughout this thesis, the Drools Rule Language (DRL) is used as a well-known language for representing rules in a rule engine. Second, all other concepts (i.e., meta-data) are translated into API calls. Furthermore, the translated rules from the first step are wrapped into valid API calls (e.g., createRule) as well.

Finally, all API calls are imported into one or more running gamification platform instances. Since this deployment step depends on a concrete implementation and does not need conceptual consideration, it is deferred to Section 8.2.1 in the evaluation chapter.

The remaining text is structured in accordance with the two-fold compilation process, i.e., the translation of entities and meta-data (Section 7.6.1) is considered separately from the compilation of rules (Section 7.6.2).

### 7.6.1 Compilation of Entities

In general, all entity definitions of GaML are compiled into methods defined by the update or query interfaces of the BEP. For each entity definition in a GaML instance, there is exactly one corresponding method call in one of the BEP's update or query interfaces.

A translation example for the point element is given in Listing 7.7. On the left-hand side, the GaML definition for a point element is given with all required parameters. On
the right-hand side, the corresponding method call available on the update interface is shown in JSON-RPC notation [105]. Note that this call is directly executable as remote service call within the gamification runtime environment and is presented in more detail in Chapter 8.

The pattern used in Listing 7.7 is used throughout the following text, i.e., the left-hand side represents a source code in GaML, the translation itself is represented by an arrow (⇒), and the right-hand side represents the target code. The text in angle brackets refers to parameters (terminals) or non-terminals which have to be further specified either by the gamification expert or through other compilation rules beyond the currently shown example.

```
1 point <ID> {  
2   name='<dName>',
3   abbreviation='<abbrv>',
4   type=<POINTTYPE>
5 }
6
1 {'id':0,'method':'createPoint',
2   params:['<ID>','<dName>','<abbrv>','<POINTTYPE>']}
7
Listing 7.7: Translation example for the point entity
```

The same translation principles also apply to other L1 concepts that are described in Section 5.2.1. For instance, Listing 7.8 presents the compilation of the badge concept into the corresponding API method. Note that the image is passed to the method call as argument, i.e., to upload the image to the runtime system.

```
1 badge <ID> {  
2   name='<dName>',
3   description='<description>',
4   image='<img>',
5   hidden=<hddn>
6 }
7
1 {id:0,method:'createBadge',
2   params:['<ID>','<dName>','<description>','<img>','<hddn>']}
8
Listing 7.8: Translation example for badge entity
```

As presented in Chapter 5 some of the L1 concepts (e.g., badge, points, or skill concepts) directly allow the definition of when clauses or rules to express the circumstances under which the defining concept can be achieved. Despite the fact that rule translations are discussed in Section 7.6.2, Listing 7.9 presents the result after all sub-compilations have been processed.

```
1 badge <ID> {  
2   name='<dName>',
3   description='<abbrv>',
4   image='<img>',
5   hidden=<hddn>
6 when player {  
7     <PlayerConditions>
8   }
9 }
10
1 {id:0,method:'createBadge',
2   params:['<ID>','<dName>','<description>','<img>','<hddn>'])
3 {id:1,method:'createRule',
4   params:['GRID1',
5     'Avatar($aid : avatarID)\n <
6     PlayerConditions>',
7     'updateAPI.addBadgeToAvatar(
8     $aid, <ID>);'}
9 }
10
Listing 7.9: Translation example for badge entity including an optional when clause
```

For practical implementation, however, this possesses some limitations and is solved differently in the evaluation.
7.6 Compilation Procedure

In this example, the badge compilation shown in Listing 7.8 is extended by an optional when clause determining the conditions under which the badge can be received by a player.

Again, the entity itself is translated into the corresponding create call of the BEP. Moreover, there is a second call createRule which contains the rule’s generated ID (GRID1), condition, and consequence as strings. Within the gamification runtime environment this rule is assembled to a DRL rule shown below, i.e., the individual arguments of the JSON-RPC call are put into placeholders of a generic rule template.

```plaintext
rule 'GRID1'

when

Avatar($aid : avatarID)
<PlayerConditions>

then

updateAPI.addBadgeToAvatar($aid, <ID>)
```

Listing 7.10: Rule assembled from Listing 7.9

For the remaining text, all rule translations are presented directly in the target language for the sake of readability. However, all rules are eventually wrapped into API calls as shown in Listing 7.9.

7.6.2 Compilation of Rules

The translation of GaML rules into executable rules for the gamification runtime environment can be divided into two parts, namely the compilation of the When and Then non-terminals of the source language. Coherently, this section considers the condition parts first. The corresponding consequences are explained afterwards.

7.6.2.1 Rule Conditions - General Elements

In Chapter 5 the conditional parts of GaML have been categorized into three classes, i.e., conditions that may apply to players and their avatars only, to teams only, or both of them. This structure is retained in the following text starting with conditions that apply to both entities first.

As a preliminary consideration, it has to be recalled that both, avatars and teams, can be made explicit through a variable binding in GaML (cf. Chapter 5). The compiler has to deal with these variables bindings. More precisely, each condition in a GaML rule refers to one so-called upper entity, i.e., either a team or an avatar, to signify if this condition refers to groups or individuals respectively. Listings 7.11 and 7.12 refer to example translations.

```plaintext
when player <pVariable> {
<PlayerConditions>
}
```

Listing 7.11: Example translation of an outer player to avatar entity

```plaintext
when team <tVariable> {
<TeamConditions>
}
```

Listing 7.12: Example translation of an outer team entity
In case the source’s condition refers to a player, the target language uses the currently selected avatar instance of that player. This is consistent with the assumption that the avatar is considered as runtime concept since a player may have multiple virtual representations in the game (cf. Chapter 4). Therefore, the gamification expert can write the rules in an avatar agnostic manner and, thus, may abstract from concrete runtime instances of the player. In case the source’s condition refers to a team, the compiler generates the target code for the same runtime entity.

Furthermore, the terminals `<pVariable>` and `<tVariable>` might be used as entity binding ID. This ID is either taken explicitly from the GaML definition or enumerated implicitly by the compiler if no ID is given. Those IDs are maintained by the compiler in a symbol table in order to allow for nested entity bindings within one rule. A simple example for the variable resolution is given in Listing 7.13.

```plaintext
when player $pid {
  <PlayerConditions>
} then { give 1 XP to $pid }
```

Listing 7.13: Simple example for variable binding

Here, the player’s current avatar is explicated through the ID `$pid`. This ID is used in the consequence to explicitly express that the outer avatar receives the reward. In this example, it would be semantically equivalent to omit the `to` clause in the GaML source code as well as the variable definition of the condition since the compiler would generate an implicit variable instead.

A more complex example is shown in Listing 7.14. Here, multiple entities are nested having different conditions. In particular, this example refers to a condition where some player issues a `TestAction`, has more than 10 `TestPoints` in total, has 50% luck, and belongs to a team where at least one player (the player himself or another teammate) has more than 20 `TestPoints` in total. If the condition is fulfilled, the utmost player receives two, all players of the team three, and the inner player four `TestPoints`.

```plaintext
when player $p1 {
  did useraction TestAction
  and has point TestPoint , SUM > 10
  and has luck P(0.5)
  and belongs to team $t1 which {
    has player $p2 who {
      has point TestPoint , SUM > 20
    }
  }
} then {
  give 2 TestPoint to $p1
  give 3 TestPoint to $t1
  give 4 TestPoint to $p2
}
```

Listing 7.14: Complex example for variable binding

Again each entity has been explicated through an ID which is used in the corresponding consequence to give each entity its defined reward. However, removing the `to` clauses or the explicit variables from this example, would not yield a semantic equivalent anymore and even would lead to a wrong outcome. In fact, the resulting switch

\[\text{Broken references in the consequences are avoided before compile-time through static semantics.}\]
to implicit variable bindings generated by the compiler implies that the utmost entity $p1$ would receive all three consequences.

Given these preliminary considerations, the translation of each conditional element is explained below based on the structure of Chapter 5.

**Boolean Constraints (E12a)** All single conditions can be correlated with standard Boolean operators (see Figure 5.4). These Boolean expressions are directly used for the target DRL representation. While GaML defines all operators, except logical “not”, with infix notation, the compiler generates them for prefix notation. Furthermore, logical “and” is implicitly assumed in DRL between consecutive conditions. Correspondingly, Listing 7.15 shows an exemplary translation:

```plaintext
when player {
  <Condition1>
  and <Condition2>
  and not <Condition3>
  or <Condition4>
}
```

Listing 7.15: Example translation of Boolean operators

**General References (E12b, E12c, E12d)** This category subsumes conditional elements which are translated in a similar manner. This applies to the conditionals of badges (E12b), levels (E12c), and items (E12d). Listing 7.16 shows one example translation per element.

```plaintext
when player {
  has badge <B1>
  and has level <L1>
  and has item <I1>
}
```

Listing 7.16: Example translation of general reference conditions

All conditions are translated into calls against the BEP’s query interface. The target template for these calls is similar across all elements, i.e., the method name includes the upper entity class it belongs to (e.g., avatar), the class of the condition element (e.g., badge). Furthermore, the method’s parameters take the variable binding ID of the upper entity class (here, the implicitly generated $aid$), and the ID of the referenced condition element (e.g., <B1>.

Each call is embedded into the `eval` function, a specific DRL construct to evaluate arbitrary expressions. Here, it is used to validate, if the BEP’s response equals to `true`. Note that this condition is also applicable to the entity team. For this case, all method names of the BEP’s query interface contain the same-named term. This applies for these conditions as well as additional ones described below.

**Mission Condition (E12e)** In contrast to the former conditions, the mission element uses the mission’s status which has to be reflected as an additional parameter in the generated target code (Listing 7.17). Upon compilation, the specific enumeration types
introduced in Section 7.4.1 are used to determine the mission state. Finally, this example shows that if no mission status is specified, the completed status is assumed by default.

Listing 7.17: Example translation of several mission conditions

**Skill Condition (E12f)**

The condition whether the entity has a skill, optionally at a particular level or activated, is translated as presented in Listing 7.18.

Listing 7.18: Example translation of skill condition

In all cases, the avatar's current state is retrieved from the BEP and the response is compared either with the result true or the required level <lvl>. Note that the active clause is part of the method's name rather than an additional parameter.

**Point Condition (E12g)**

The point condition examines, if the aggregated quantity of points compared to a threshold is fulfilled by the entity. The abstract translation rule executed by the compiler is presented in Listing 7.19.

Listing 7.19: Example translation of point condition

Here, the additional parameters are influencing the operator and the right operand within the DRL eval function. While the point ID, the aggregation method, and the operator refer directly to terminals, numeric expressions (NUMEXPR) have be translated further as shown later on.
7.6 Compilation Procedure

**Luck Condition (E12h)**

Similarly, Listing 7.20 shows the translation of the luck condition. However, the BEP is not required in this case. Instead, a helper class is used to generate a pseudo-random number between 0 and 1 which is compared numerically against a defined threshold\(^8\).

```plaintext
1 when player {  
2   has luck P(<NUMEXPR>)  
3 }  
```

Listing 7.20: Example translation of luck condition

**Location Condition (E12i)**

The location condition examines, if the entity is at a particular location with regards to longitude and latitude. Listing 7.21 shows an example translation.

```plaintext
1 when player {  
2   is at location(lat <comp1> <lat>, long <comp2> <lon))  
3 }  
```

Listing 7.21: Example translation for location condition

The user’s or team’s current position is determined over the standardized event `movement` (Section 7.5.2) which is automatically available, if at least one rule requires a current location in its condition. The EPA provides a procedure to match the target position with the source position. This procedure must allow for a configurable precision at runtime\(^9\).

**Leader Condition (E12j)**

Finally, Listing 7.22 shows the translation of the leader condition. In this case, the BEP’s query interface is used to retrieve the latest leaderboard as provided in the corresponding `GaML` specification.

```plaintext
1 leaderboard Board1 {  
2   name = <LID>, point = <PID>,  
3      aggregation = <AGG>,  
4      order = <ORDER>  
5 }  
```

Listing 7.22: Example translation of leader condition

\(^8\)Note that `GaML` does not allow to specify the underlying probability distribution function. Therefore, numbers are drawn from an uniform distribution currently.

\(^9\)Note that `GaML` does not allow to define a particular radius around the specified location. However, this would comprise a simple extension to `GaML`’s grammar and compiler and is subject to future work.
Here, the result from the leaderboard method is explicated through the variable $value$ which is used in the subsequent eval statement within DRL to check the value range. Recalling the data model (Section 7.4.1), it is important to note that leaderboards are transient data structures. Therefore, they are uniquely identified through the combination of point ID and aggregation method, i.e., even if an entity may participate in multiple leaderboards, those can be referenced precisely.

### 7.6.2.2 PLAYER CONDITIONS

After the presentation of conditions that apply equally to avatars and teams, this section is concerned with avatar- or player-specific conditions. More precisely, the player construct in GaML comprises three unique language constructs, namely the examination of virtual roles, the belongingness to a team, and the issuing of user actions for oneself or in relationship to other players.

**Role Condition (E12k)** Listing 7.23 shows the general rule for translating the role condition construct, i.e., if the player’s current avatar has a particular role.

<table>
<thead>
<tr>
<th>Code</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>when player {</td>
<td>Avatar($aid : avatarID)</td>
</tr>
<tr>
<td>has role &lt;RID&gt;</td>
<td>eval(queryAPI.hasAvatarRole($aid, '&lt;RID&gt;'))</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

Listing 7.23: Example translation of role condition

**Team Belongingness (E12q)** This language element is used to identify, if a player’s avatar belongs to a particular team which is in a certain state. For this purpose, GaML defines an avatar- or player-specific construct whose general translation is presented in Listing 7.24.

<table>
<thead>
<tr>
<th>Code</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>when player {</td>
<td>Avatar($aid : avatarID)</td>
</tr>
<tr>
<td>belongs to team &lt;tVariable&gt;</td>
<td>eval(queryAPI.hasAvatarRole($aid, '&lt;RID&gt;'))</td>
</tr>
<tr>
<td>which &lt;TeamOr&gt;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

Listing 7.24: Example translation for player-team relationships

In this case, the team expression of GaML is directly translated into a DRL statement which joins all teams with the respective avatar. The resulting list can be further pruned with the `which` statement where the non-terminal `TeamOr` is compiled as presented in Section 7.6.2.3.

**User Actions and Events (E12l-E12p)** Only players are allowed to issue user actions in the system (Section 5.2.2). The GaML statements `did <EventDef>` and another player `did <EventDef>` refer to the issuing of events for oneself or other players (opponents or teammates). Thus, the following text describes the non-terminal `EventDef` as a general building block to process events.

Listing 7.25 shows a simple example of an event condition referring to an event type (E12l) and two property constraints (E12m). It is shown that the event’s type is directly translated into the `type` property. Additional constraints on the event’s payload are converted based on their defined type in GaML, e.g., a number is explicitly converted into an integer representation or text is retained as string representation in DRL. In
7.6 Compilation Procedure

addition, the event is automatically joined with the enclosing entity as defined in the GaML source, in this case, the avatar. Finally, the event is consumed from particular entry-points. Hereby, the event’s class determines which entry-point is selected. For example, external events or user actions are consumed from the eventstream as shown in Listing 7.25 and internal events from the internalstream entry-point (not depicted).

```
useraction <EVENTTYPE> {
    properties {
        <PROP1> : Number,
        <PROP2> : Text
    }
}
```

Listing 7.25: Translation of event type (E12l, E12m)

Listing 7.26 shows a translation example for the joining of events for the same and another player. In this example, the player has to consecutively issue two events of the same type, i.e., the second event has to arrive after the first one. Although the events’ types are equal, the customer’s ID in the events’ payloads has to be different. Furthermore, the overall condition becomes true, if any other player, additionally, issued an event of the same type and for the same customer ID as in the first event. All operators, i.e., the two join (E12n) and the after temporal (E12o) operators can be directly translated into DRL shown in Listing 7.26.

```
when player {
    did <VAR> : useraction < EVENTTYPE>, <PROP1> < COMPARATOR> <NUMEXPR>, <PROP2> = <EQUALITYTYPES>
}
```

```
when
    Avatar($aid : avatarID)
    <VAR> : EventObject(type=='<EVENTTYPE>', Integer. parseInt(data['<PROP1>']) < COMPARATOR> <NUMEXPR>, data ['<PROP2>']='<EQUALITYTYPES>', avatarID== $aid) from entry-point eventstream
```

Listing 7.26: Translation of event joins and temporal operators

Additionally, events which participate in at least one join condition require the abstraction of this event into an interval event. Otherwise it would be retracted automatically after the evaluation cycle and the join of events with different timestamps may fail.

```
when player {
    did evtl : useraction < EVENTTYPE>
    and did evt2 : useraction < EVENTTYPE>, customer <> evtl.customer, this after evtl
    and another player did evt3 : useraction <EVENTTYPE>,
        customer == evtl.customer
}
```

```
when
    Avatar($aid : avatarID)
    evtl : EventObject(type='<EVENTTYPE>', duration>0, avatarID==$aid) from entry-point internalstream
```

```
evt2 : EventObject(type='<EVENTTYPE>', duration>0, data['customer']==evt1.data
    ['customer'], this after evtl, avatarID==$aid)) from entry-point internalstream
```

```
evt3 : EventObject(type='<EVENTTYPE>', duration>0, data['customer']==evt1.data
    ['customer'], avatarID!=$aid) from entry-point internalstream
```

```
when
    Avatar($aid : avatarID)
    <VAR> : EventObject(type=='<EVENTTYPE>', duration>0, avatarID==$aid) from entry-point eventstream
```

```
evt2 : EventObject(type='<EVENTTYPE>', duration>0, data['customer']==evt1.data
    ['customer'], this after evtl, avatarID==$aid)) from entry-point internalstream
```

```
evt3 : EventObject(type='<EVENTTYPE>', duration>0, data['customer']==evt1.data
    ['customer'], avatarID!=$aid) from entry-point internalstream
```
Therefore, for such events additional rules are generated which add a configurable duration to the events and insert those into the internal entry-point called `internalstream`. In fact, there is exactly one abstraction rule generated per event type when the event in question participates in at least one join condition. This rule reacts on the incoming point-in-time event and creates interval events out of them by adding a duration to a copy of the event (Listing 7.27).

```java
when
  old_evt : EventObject(type=='<XYZ>', duration==0) from entry-point eventstream
then
  EventObject interval_evt = new EventObject(old_evt);
  interval_evt.setDuration(<GlobalExpirationTime>);
  entryPoints[‘internalstream’].insert(interval_event);
```

Listing 7.27: Creation of an interval event using global expiration time

There are two different ways to influence and design the duration for the event. First, a global expiration time can be configured in the gamification runtime environment. Without modifications to the GaML instance shown above, this global time is added to the respective events as shown in Listing 7.27. Consequently, the gamification or IT expert has to define a maximal time period that applies equally across all events, i.e., the event type with the longest necessary duration defines the duration for all other events. Although this mechanism is convenient, it is not very efficient with regards to resource utilization.

Therefore, the second approach allows to define a maximal duration per condition. In this case, the IT expert may utilize GaML's time window operator (E12p) to specify the event's duration. Based on the specified time window, the event's duration is selected and inserted instead of the global expiration time (Listing 7.27). If multiple time windows are specified for the same event type, the maximum duration is selected.

Consequently, the resulting target code in Listing 7.26 references the events from the entry-point `internalstream` rather than the external `eventstream`. Moreover, the event selection checks the duration attribute for being greater than zero, i.e., the rule only reacts on interval based events. The IT expert is responsible for selecting appropriate window sizes at design-time in conjunction with the business logic, i.e., he or she has to keep in mind that events are retracted after the specified time windows and, thus, cannot be joined correctly anymore.

The same procedure applies to event aggregation, i.e., where events or their payloads are aggregated over specific time windows as shown in Listing 7.28. Again, events referring to a particular time window are consumed from the internal stream after they have been abstracted to an interval event as described above.
7.6 Compilation Procedure

Listing 7.28: Example translation for event aggregation over time windows

```java
when player {
    did useraction <EVENTTYPE>,
        <AGG>({<PROPERTY>} < COMPARATOR> <NUMEXPR> within <TIME>)
} =
when Avatar($aid : avatarID)
    Number(doubleValue <COMPARATOR>
        > <NUMEXPR>) from
    accumulate(
        evt1 : EventObject(type=='<EVENTTYPE>',
            $prop:Double.parseDouble(data['<PROPERTY>']), $aid
            ==avatarID) over
            window:time(<TIME>) from
            entry-point internalstream, <AGG>($prop))
```

Finally, the lastsFor operator requires a more complex translation as presented in Listing 7.29 (source) and Listing 7.30 (target).

Listing 7.29: Example source code for lastsFor operator

```java
useraction <eventName> {
    properties { <joinProperty>:Decimal, customer:Number }
    inverseEvent { name = <deleteEvent>, joins on=<joinProperty> }
}

when player {
    did evt1 : useraction <eventName>, customer = '3', lastsFor=<timePeriod>
} <Then>
```

The GaML source defines an event and its inverse delete event having a customer ID and join value as attributes. This event is used by a condition where the customer ID has to be 3 and the event should last for the specified time period within the system, i.e., the inverse delete event must not occur in between. Otherwise, the condition is not satisfied anymore. This mechanism might be utilized, for example, to prevent cheating scenarios where the user constantly creates and deletes content in and from the system merely for the purpose of getting more rewards.

Listing 7.30 shows the resulting target code. Again, the first rule is generated based on the idea that the point-in-time event has to be abstracted into an interval event (RID1) first, i.e., omitting all additional constraints and operations on that event (e.g., checking the customer ID). As explained for events in join conditions or under time windows, RID1 generates a copy of the event and adds the duration defined by the lastsFor operator. The rule RID3 reacts on this newly created interval event and delays the rule consequence as defined by the duration property.

In the meantime, the user may issue the inverse event, i.e., the delete event, as external one. This activates rule RID2 which immediately leads to a retraction of the interval event created by RID1, thus, canceling the activation of rule RID3. The join of both, the original and the inverse event, in rule RID2 is accomplished using the joinsOn attribute of the corresponding GaML definition.

```java
rule 'RID1'
when
    Avatar($aid : avatarID)
        evt1 : EventObject(type=='<eventName>', duration=0, $aid==avatarID)
        from entry-point eventstream
then
    EventObject obj = new EventObject(evt1);
    obj.setEventDuration(<timePeriod>);
```
entryPoints['internalstream'].insert(obj);

rule 'RID2'
when
   Avatar($aid:avatarID)
evt1:EventObject(type=='<eventName>', duration>0, $aid==avatarID) from entry-point internalstream
   evt2:EventObject(type=='<deleteEvent>', $aid==avatarID, data['<joinProperty>']==evt1.data['<joinProperty>']) from entry-point eventstream
then
   retract(evt1); retract(evt2);

rule 'RID3'
delay(expr:$duration;)
when
   Avatar($aid:avatarID)
evt1:EventObject(type=='<eventName>', $duration:duration>0, avatarID==$aid) from entry-point internalstream
then
   EventObject obj = new EventObject(evt1);
   obj.setType('expired_'+<eventName>);
   obj.setEventDuration(<GlobalExpirationTime>);
   entryPoints['internalstream'].insert(obj);

rule 'RID4'
when
   Avatar($aid:avatarID)
evt1:EventObject(type=='expired_'+<eventName>, avatarID==$aid, data['customer']=='3') from entry-point internalstream
then

Listing 7.30: Example target code for lastsFor operator

If, however, RID3 is executed successfully, another event (‘expired_’+<eventName>) as copy of the interval event is created to signify that the interval event was success-
fully delayed. This new event is again marked as an interval event either based on the
globally configured expiration time or based on a specified time window in GaML. This
activates the last rule RID4. In addition, all omitted constraints and operations on the
original event are now applied to the abstracted event. This is necessary to allow for
multiple lastsFor operators within the same rule but with different timings, i.e., all event
conditions having a lastsFor condition are required to go through their own three pre-
processing rules until all of them or jointly processed in the rule RID4. Consequently,
the lastsFor operator has the highest precedence in GaML.

7.6.2.3 TEAM CONDITIONS

Team Belongingness (E12q)  The only team-specific condition is to check, if one or
more particular avatars belong to the team in question and, optionally, are in a particular
state to prune the result. Listing 7.31 shows an exemplary translation from GaML to DRL.

when team {
   has player $pid who < PlayerOr>
}

when Team($tid : teamID)
   (Avatar($pid : avatarID, teams.contains($tid))

Listing 7.31: Example translation of player or avatar belongingness
In this example, the avatar’s belongingness is checked using the entities teams property. In addition to the membership, the avatar may have to comply with some arbitrary condition as specified in the who clause of GaML. Here, all above describe player-specific or player-independent condition clauses might be used.

This concludes the translation procedure for rule conditions. Subsequently, the compilation of rule consequences is described in more detail.

7.6.2.4 RULE CONSEQUENCES

As shown in Chapter 5, possible consequences include new events (E13a), points (E13b), notifications (E13c), narrative messages (E13d), badges (E13e), items (E13f), missions (E13g), and skills (E13h). Selected code generations are presented below.

**Event Consequences (E13a)** Listing 7.32 present a translation example for event consequences. The then clause in GaML creates a new event type with exactly one key value pair. Furthermore, the event is issued for an entity identified through its ID (<EID>), i.e., the corresponding condition has to make the player (avatar) or team explicit using a variable.

Within the target code, those information is mapped onto Java code. This includes the creation of a new event object and using the setter methods to set the event type, the avatar’s id, and all key value pairs. Finally, the event is inserted into the internal entry-point internalstream from where it can be consumed by other rules.

```java
1 then {
2   create event <EVENTTYPE>(<PROP1>=<VAL1>) to <EID>  
3   newEv.setType(<EVENTTYPE>);
4   newEv.setAvatarID(<EID>);
5   newEv.put(<PROP1>,<VAL1>);
6   entryPoints["internalstream"].insert(newEv);
}
```

Listing 7.32: Example translation of event consequences

**Other consequences (E13b-E13h)** All other consequences follow two particular patterns, namely a pattern for giving and a pattern for deleting an instance of the concept in question.

Listing 7.33 shows a general translation example for assignment consequences, i.e., where the current avatar of the player is assigned a new instance of the respective entity. Again, the same also applies directly to consequences for team entities. In this case, there are methods in the BEP’s update interface available whose method name contain the term Team instead of Avatar. Here, it depends on the type of the referenced <EID> which entity is addressed by the method’s name. If no to clause is specified in GaML the class of the utmost entity is used (see Section 7.6.2.1).
then {  
give <AMNT> <PID> to <EID>  
give badge <BID> to <EID>  
give mission <MID> to <EID>  
notify '<TEXT>' to <EID>  
narration(name='<dName>', description='<DESC>', recipient=<EID>)  
}  
⇒  
updateAPI.givePointsToAvatar(<PID>, <EID>, <AMNT>);  
updateAPI.addBadgeToAvatar(<BID>, <EID>);  
updateAPI.addMissionToAvatar(<MID>, <EID>);  
updateAPI.notifyAvatar(<EID>', '<TEXT>');  
updateAPI.narrateAvatar(<EID>, '<dName>', '<DESC>');

Listing 7.33: Example translations assignment consequences

The same structure applies also to the deletion of instances within consequences. Listing 7.34 presents example translations for selected elements. In contrast to the assignment of points or badges, the methods’ names include the term remove rather than give or add.

then {  
remove <AMNT> <PID> from <EID>  
remove badge <BID> from <EID>  
remove mission <MID> from <EID>  
}  
⇒  
updateAPI.removePointsFromAvatar(<PID>, <EID>, <AMNT>);  
updateAPI.removeBadgeFromAvatar(<BID>, <EID>);  
updateAPI.removeMissionFromAvatar(<MID>, <EID>);

Listing 7.34: Example translations of deletion consequences

7.6.3 COMPILATION OF TERMINALS

The last sections deferred the translation of terminals. Some of the terminals (e.g., COMPARATOR, ID, STRING, or AGGREGATION) can be directly reused in the DRL, i.e., there is no additional translation necessary. Other terminals such as NUMEXPR or TIME, however, require a more complex translation procedure and are presented below.

For example, every terminal of type TIME has to be translated into a corresponding millisecond representation as shown in Listing 7.35.

static def int toMilliseconds(TIME time) {
  time.value * 1000 * switch time.name {
    case 'w': 7*24*60*60  
    case 'd': 24*60*60  
    case 'h': 60*60  
    case 'min': 60  
    default: 1  
  }
}

Listing 7.35: Compilation of TIME terminal

As another example, Listing 7.36 shows the translation of numeric expressions into its corresponding DRL part. This generation has been used, for example, in Listing 7.28 to convert the numeric expression into to the respective expression of the target language to compare its results against an aggregation threshold.
7.6 Compilation Procedure

```java
static def dispatch generateCode(NUMEXPR expr) {
    val it = expr
    if (operand1 != null) {
        "'"+operand1.generateCode+"IF operator != null\" operator \" operand2. generateCode\" ENDIF'"+
    } else if (innerOperand1 != null) {
        "'"+(innerOperand1.generateCode+"IF innerOperator != null\" innerOperator \" innerOperand2.generateCode\" ENDIF)+"' +
        "'IF operator != null\" operator \" operand2.generateCode\" ENDIF'"+
    }
}

static def dispatch generateCode(NUMERICOPERAND operand) {
    val it = operand
    if (refOp != null) {
        "$refOp.eventRef.name\"refOp.^var.name ?: \"aid\""
    } else if (d != 0d) {
        d.toString
    } else if (^int != 0) {
        ^int.toString
    } else {
        '0'
    }
}
```

Listing 7.36: Translation of numeric expressions

### 7.6.4 SYSTEMATIZATION

This chapter introduced the gamification runtime environment in two main parts as shown in Figure 7.9. First, a generic hybrid architecture comprising a database (Business Entity Provider) and rule engine with event processing capabilities (Event Processing Agent) has been proposed based on a discussion of the non-functional requirements of Chapter 4 and their associated trade-offs.

Second, since all parts of the hybrid architecture have been described as domain-independent components, Sections 7.4 and 7.5 provided gamification-specific adaptations. It has been argued that those specific adaptations (e.g., syntax and semantics of APIs, rule language model) comprise the platform specific model used by the gamification runtime environment.

![Figure 7.9: Summary of the conceptual translation procedure](image)

Afterwards, the gap between the GaML model developed in Chapter 5 on the one hand and the PSM and runtime environment on the other hand, has been bridged by a model-to-model translation, i.e., from GaML into executable code for the gamification runtime environment.

However, with regards to Figure 7.9, two aspects have not been considered. First, the generation and implementation of the parser to create the AST has been omitted.
Second, the transformation of the API calls (i.e., PSM) into the actual runtime system was not presented. Since both aspects are very implementation-specific they are considered in Section 8.2.1 of the evaluation.

Furthermore, the remaining thesis described the evaluation of the developed artifacts with regards to their feasibility and non-functional requirements. For example, this comprises the presentation of the implemented gamification runtime environment on a fine-grained level as well as their evaluation with regards to requirements such as performance, reusability, or flexibility. Additionally, GaML is evaluated with regards to the stated design objectives.
8 EVALUATION

In this chapter, the contributions of this thesis are evaluated with regards to feasibility, implementation, efficiency, and applicability. First, the applications where the gamification platform was utilized to implement real and productive scenarios are introduced. Based on these applications, the various artifacts of this thesis are evaluated. Second, the implementations of GaML and the generic runtime environment for gamification are validated. Third, the applicability of GaML with regards to its design goals (Section 5.1) is evaluated. Finally, the overall benefit of the gamification platform and its methodology and tools is evaluated in an end-user test.

8.1 APPLICATION SCENARIOS

In the following, five real and productive business applications are introduced. These applications are a selection of all cases where gamification was introduced using the concepts provided by this thesis. They have been selected to examine the different aspects and contributions (e.g., reusability in different domains) of this thesis. Throughout the evaluation, all of them are considered in more detail.

8.1.1 SAP TWOGO

This application covers the gamification of a carpooling system, where people can share rides with each other. The already gamified solution is deployed in a large company with approximately 8,000 regular users. In contrast to traditional commuting applications, users are automatically matched once they posted their ride intents. However, in a corporate context the problem exists that people need to be motivated to share their car as a driver which is often perceived as being less flexible and inconvenient.

In accordance with the gamification process (Chapter 3), the first step was to analyze the business process as well as its potential users. Therefore, a qualitative analysis has been conducted to find out what motivates the users of the application in general. This analysis yielded that people especially use the application to be ecologically sustainable, to extend their social network, to arrive at work with less stress, or perceive a smaller likelihood of accidents. Furthermore, it was observed that there were many passengers willing to share rides, however, drivers do not participate equally often as they are less flexible on the one hand and have to accept disadvantages in corporate tax regulation with their company cars on the other hand.
After a thorough requirements analysis, the design phase started as proposed by the overall development process. Multiple iterations were taken to create and design the gamification concept and discuss it with all stakeholders involved in the project (e.g., product owner, architect, managers, and business experts). Furthermore, a design contest was conducted where nine teams proposed very different gamification concepts based on the findings of the requirements phase. All these concepts were synthesized into an overall concept for the carpooling application.

The major parts of the gamification include mechanisms to reinforce and uncover the observed intrinsic motivations such as collecting points for saving carbon-dioxide or meeting people. In particular, the carbon-points can then be used to grow a virtual tree which, once completed, is planted in reality to reinforce the need of being ecologically sustainable further. Additional badges were introduced for special situations, e.g., sharing a ride with the company’s CEO or to fetch a colleague after 10pm from a particular bar. A virtual story was invented to on-board new users quicker by fostering curiosity. For experienced users, collective goals such as saving one million kilograms of carbon-dioxide were proposed. Moreover, experienced users may select from a set of avatar roles which makes them responsible for certain aspects in the solution (e.g., to announce new goals for all users of the solution).

The implementation of this overall design was also divided into multiple iterations, i.e., the first iterations were concerned with the implementation of basic game mechanics and simple operational rules. Subsequent iterations added the remaining concepts then. The examples shown in this thesis refer to the early iterations as its results are in a productive state. The concrete rules are shown later when the integration scenario is elaborated in more detail (Section 8.3). Figure 8.1 shows the visualization of points in the user interface of the target carpooling system. Nevertheless, other concepts and rules have been introduced over the system’s lifetime without changing the host application. This indicates that the platform provides the desired degree of flexibility with regards to design changes to a running gamification concept.

Figure 8.1: Example screenshot from the SAP TwoGo gamification project
8.1 Application Scenarios

8.1.2 SAP NETWORKING LUNCH

This application covers the gamification of a networking application which automatically matches users, who typically do not know each other, for lunches, coffee breaks, or other networking opportunities. Before starting the gamification process, this application was used by approximately 10000 users (around 7000 male and 3000 female users) where 1000 users use the application at least once per day. The gamification process was applied to investigate the business model and requirements in order to design a comprehensive gamification concept.

The final concept foresees different missions that should engage users to explore and use all aspects of the application. Gamification is used to reward various user actions, e.g., the user gains points for each added colleague or accepted meeting. If a specific amount of points has been achieved, the user completes a particular mission, receives a badge, and is assigned another mission. For example, after accepting the first meeting, the user completes the Accept First Meeting mission, receives the corresponding badge and gets the next mission (e.g., Host a Group Meeting). In order to support latent psychological factors such as curiosity or surprise, some of the badges are hidden by default. Figure 8.2 shows an example of the networking application and some of the badges users may earn along their usage and participation.

Figure 8.2: Example screenshots from the SAP Networking Lunch gamification project
8.1.3 SAP FINANCIAL FACT SHEET

In contrast to the previous applications, this scenario covers the gamification of a standard economical business process from an ERP system, namely a financial fact sheet application that is used for generating leads. Furthermore, this scenario specifically aims at mobile application where the gamification should be introduced.

With regards to the gamification concept, employees should be engaged to generate more leads by reminding customers who have not paid their bills yet. Therefore, users may gain several points for successful reminders or delegating tasks to their colleagues by writing notes. Users gain more points the higher the total payment is. Further elements include notifications, time pressure, and competition. For example, bonus missions are unlocked that allow to acquire more points when a customer pays within a given time period. Moreover, gamification elements have been used to onboard new users quickly by guiding them through the application’s capabilities. For instance, gamification goals encourage the users to find and explore all available views and possibilities of the application. Figure 8.3 shows the integration of the gamification aspect into the existing mobile application.

Moreover, this scenario also requires a more complex integration with the existing system landscape on the back-end side, i.e., the gamified mobile application is only the final representation for the user. However, in the background the gamification platform has to be integrated with further systems such as an ERP system that holds the relevant business data and a mobile application management platform (e.g., SAP Mobile Platform) that offers additional enterprise-related services for mobile applications. Because of its additional complexity with regards to integration, this scenario is selected for Section 8.6 to validate the benefits of the thesis’ approach.

Figure 8.3: Example screenshots from the SAP Financial Fact Sheet gamification project
8.1.4 SOCCER TRAINING APPLICATION

This application scenario is again different compared to the aforementioned applications with regards to two main aspects.

The first different aspect is that the host application itself qualifies as a serious game rather than gamification according to the respective definitions. As such the application is an ensemble of several mini-games which are used to train mental soccer skills such as reaction times as well as faster perception and assessment of different situations. Figure 8.4 shows one of these mini-games, namely BrainShift. All games have been developed together with the chief-psychologist of a soccer club playing in the German Bundesliga and, thus, were designed based on current psychological findings to train the players’ perceptions and reactions in a gameful experience. Each mini-game is composed of several game levels which increase in difficulty. In each game level, ten rounds can be played with the goal to reach as many points as possible. Based on the achieved score the player either has to repeat the level or qualifies for the next one.

![Example screenshot from the Soccer application with gamification](image)

The gamification runtime system is used to store the achieved score and generate aggregated structures such as avatar levels or leaderboards. Trainers then can use the platform’s analytical features to analyze players’ development over time or to compare their performances. One special requirement was that the games should be also playable offline (e.g., while traveling). Therefore, the game was completely developed for the mobile client and only synchronizes with the gamification runtime environment when an online connection is available. Consequently, the runtime is used as Achievement System (AS) in accordance with the derived solution classes of Chapter 6. This leads to the second different aspect of this application scenario.

After deployment of the first version, it was recognized that the integration and customization is not flexible enough with regards to design changes. For example, the soccer trainer was not able to configure and maintain the point calculation rules for the different levels without a recompilation and redistribution of the mobile application. Therefore, the second version of this application utilizes the features of the gamification...
tion platform further by introducing business and event processing rules as proposed in this thesis to allow a more flexible design at runtime. With regards to the solution classes, the new requirements transformed the gamification runtime from an AS into a Generic Gamification Platform (GGP). This demonstrates that the proposed runtime environment is highly flexible and reusable even for very specific types of applications.

After multiple iterations within the gamification process, this application is now used as a daily exercise in addition to traditional ones.

### 8.1.5 CHIO EVENT APPLICATION

This application concerns the gamification of the CHIO (Concours Hippique International Officiel), which is an international horse sport event that takes place annually and is visited by around 360,000 visitors [56]. A mobile application has been developed to engage and entertain visitors. The application comprises quizzes, votings, leaderboards, and other interesting statistics for visitors as well as the organizers.

![Figure 8.5: Example figures from the CHIO gamification project](image_url)
Here, the gamification runtime has been used as an achievement system. Visitors directly interacted with the gamification runtime over their smart phone applications. For example, they could vote for their favorite horse rider or participate in quizzes testing their knowledge on the riders in particular and the event in general. The riders with the most votes were shown on publicly available scoreboards in the tournament area. Figure 8.5 depicts an example screenshot of the mobile application on the left hand side and the public scoreboard showing the most popular horse riders on the right hand side.

Overall, this application has been used by around 5300 users\(^1\) to complete the quiz or to vote for the most popular horse riders.

**8.1.6 ADDITIONAL APPLICATIONS**

Besides the five selected examples, the developed gamification platform has been integrated or used in a variety of other application contexts such as help desk and support applications, knowledge management tools (e.g., MediaWiki), HR applications, portals, education and learning environments, or e-ticketing and loyalty scenarios. However, since these cases are either still under development or do not contribute to the validation of the thesis’ design goals, they are not discussed further.

Overall, it is argued that the quantity and quality of the applications which have been gamified using the developed platform demonstrate the desired degree of reusability. This statement is supported by the heterogeneous types and requirements of these applications. Furthermore, flexibility is considered to be high as the examples demonstrate that the platform supports the user in the adoption of design changes in multiple different application scenarios.

**8.2 IMPLEMENTATION**

Up to now, the two main artifacts of this thesis, i.e., GaML (Chapter 5) and the proposed runtime system (Chapter 7) have been conceptualized and described on a theoretical level only. Therefore, this section describes concrete implementations, i.e., software components and tools, to demonstrate the general feasibility of the theoretical concepts and to enable their practical use. In accordance with the thesis’ structure, the implementation of GaML is described first, followed by the presentation of the runtime environment’s implementation.

**8.2.1 GAMIFICATION MODELING LANGUAGE**

The implementation of GaML complies with the transformation process presented in Figure 7.8 where a parser creates an AST based on a valid GaML instance and a code generator creates code in the target language using the created AST. In the following text, the implementation of these two building blocks is described.

**8.2.1.1 PARSER**

In order to validate instances of GaML with respect to its meta-model and static semantics (Chapter 5), a textual editor has been created using xText [73] including lexer and

---

\(^1\) Statistics are based on the first event in 2013 where the application has been deployed.
8 Evaluation

parser\(^2\). Hereby, xText requires an EBNF-similar (Extended Backus-Naur-Form) grammar as .xtext file. This .xtext file complies fully to the conceptualized grammar of Chapter 5 and Appendix A. From his grammar, a recursive descendant parser is automatically generated using the ANTLR parser generator [151]\(^3\).

Furthermore, the grammar is used to generate an Ecore model which represents the meta-model or abstract syntax of GaML. Based on this model, the xText framework can automatically generate an Eclipse-based editor where IT or domain experts may write valid GaML instances. Figure 8.6 shows an example from the resulting editor.

![Carpooling example in GaML editor](image)

The example shows that GaML authors are supported through visual highlighting of the language’s keywords, code completion for terminals, and immediate presentation of syntactic and semantic errors, i.e., when the input does not comply to the language’s meta-model.

With no additional input to xText, this environment parses a well-formed GaML instance and stops after creating the internal parse tree for this instance. In order to

\(^2\)The final editor can be found in the supplementary materials of this thesis.

\(^3\)A recursive descendant parser is a type of top-down parser and, thus, allows to decide for languages of class \(LL(k)\) whether an instance belongs to the language or not. As an implication, the input grammar has been left refactored to avoid infinite left recursions.
check static semantics on the AST, the meta-model (i.e., the Ecore meta-model) can be enriched directly with the OCL expressions given in Chapter 5. Afterwards, the editor considers these additional constraints leading to the display of errors or warnings when constraints are not met.

Figure 8.7 shows a selected application of two invariants for the redeemable point semantics (e.g., Listing 5.3) which are introduced within the meta-model.

```java
class Point extends Element {
    attribute default : Boolean[];
    attribute dName : String[];
    attribute description : String[];
    attribute abbreviation : String[];
    attribute internal : Boolean[];
    property type : POINTTYPE[] { composes };
    property rules : Rule[*] { ordered composes };

    invariant RemoveImpliesRedeemable:
    self.rules._'then'.consequences.consequences
    -select(c | c<>null and c.oclIsTypeOf(PointRefCons))
    -exists(prc | prc<>null and prc.oclAsType(PointRefCons).action = 'remove')
    implies self.type.name = 'REDEEMABLE';
}

class PointRefCons {
    attribute action : String[];
    property points : NUMEXPR[] { composes };
    property point : Point[];
    property who : EntityCond[] { composes };

    invariant RemoveImpliesRedeemable2:
    (self.action='remove' and self.point<>null)
    implies self.point.type.name = 'REDEEMABLE';
}
```

Figure 8.7: OCL expressions applied in Ecore meta-model

8.2.1.2 CODE GENERATION

The code generation part concerns the translation of GaML into valid API calls for the implemented gamification runtime. As explained in the transformation process (Figure 7.8), business and event processing rules reflecting GaML conditions and consequences are generated and wrapped into API-calls as well.

Figure 8.8 shows a more concrete version of the general compilation process presented in Figure 7.8 (p. 101) considering five steps. First, GaML elements that describe master data such as points or badges are directly translated into the corresponding API calls using the JSON-RPC (Javascript Object Notation - Remote Procedure Call) representation. Second, all GaML rules are translated into the Drools Rule Language (DRL) since the core rule engine of the prototypical runtime environment is implemented using Drools. Third, the generated rules are also wrapped into API calls for the gamification runtime environment. Fourth, one Javascript Object Notation (JSON) file with all API commands plus all referenced resources of the GaML source (e.g., images for badges, levels, or items) are packaged and zipped into one consumable package. Finally, the generated ZIP file can be imported into the system using the platform's administration User Interface (UI).
The code generation has been implemented with the xText framework utilizing the Xpand template engine which allows for model-to-text translations. Input to the framework is one .xtend file which implements the code generation for each element of GaML’s meta-model. This translation procedure complies fully to the compilation concept described in Chapter 7.6.

Figure 8.9 shows a running example of the code compilation. In the left window pane, the source file CarPool.gaml can be found whose content is shown in the right window pane. Furthermore, within the left window pane, the src-gen folder contains the corresponding CarPool.zip which holds a file gp_concept.json that consists of all API calls in JSON format as well as all images referenced by CarPool.gaml. By contract, these images have to be provided in the resources folder next to the original source file.

Figure 8.9: Example of code generation within GaML editor
8.2.2 GAMIFICATION RUNTIME ENVIRONMENT

The gamification platform runtime as conceptualized in Chapter 7 has been implemented as web application for the SAP HANA Cloud platform or other JavaEE web-containers such as Tomcat. The SAP HANA Cloud platform offers hosting and management of standard Java web applications, however, does not fully comply with the official JavaEE standard as some features such as Enterprise Java Beans (EJB) or Java Messaging Service (JMS) functionality are missing by default.

In general, the entire system consists of five components delivered as WAR-files (Web Archive), namely an AchievementProvider reflecting the BEP of Chapter 7, a RuleEngine reflecting the EPA of Chapter 7, a central Gateway component for accessing the AchievementProvider and the RuleEngine, an AdminUI that allows domain and gamification experts to design, model, and maintain the gamification concept, and a GamificationWidget component which can be used to for integrating visual gamification elements into the front-end of the host application (Figure 8.10).

The internal structure of these five components as well as the overall implemented architecture is shown in the following text.

8.2.2.1 COMPONENT: ACHIEVEMENT PROVIDER

Figure 8.11 shows the internal structure and external interfaces of the AchievementProvider component. In conjunction with the conceptualized architecture of Chapter 7, this component offers a Query- and UpdateAPI\(^4\) which allow to query and update the state of the domain entities. Additional APIs provide management and retrieval functionalities for other aspects of these entities. For example, the AnalyticsAPI enables specific analytical queries, e.g., to determine engagement criteria such as the number of registered users or user actions grouped by teams, player, or timestamp. The AdminAPI can be used to manage data and meta-data that only IT experts and the administrators of the solution are allowed to change, for example, creating master data for domain entities such as badges or points. Finally, the UserConfigAPI is used to change user-specific settings such as the avatar names, privacy settings, or badge showcases. All described API components inherit from an abstract API component which defines common functionality. In particular, the Abstract API requires the JPAManager which abstracts specific database operations. The JPAManager itself requires the data model (i.e., collection of domain entities) and a JDBCInterface. The latter is also required as an external interface and is resolved and bound at runtime using the web-container’s Java Naming Directory Interface (JNDI) mechanism.

\(^4\)Excerpts for all APIs are given in Appendix B.
All APIs are exposed over a generic web-interface. In between, a generic marshalling component handles incoming Hypertext Transfer Protocol (HTTP) requests via POST or GET, dispatches the request to the respective API and serializes the result in a client-readable format, e.g., JSON. In the AchievementProvider only one concrete implementation of this generic marshalling component exists, namely, a JsonRPC marshaller that takes RPCs calls in JSON representation and reflectively calls the corresponding interfaces.

For example, the JSON GET request for player data shown in Listing 8.1 on the left side calls the corresponding method implementation of the IQueryAPI shown on the right side.

```
{ 'method':'getPlayer', 'id':1,
  params:['some.player@domain.com']}
public Player getPlayer(String uid);
```

Listing 8.1: Mapping of JSON-RPC call to corresponding Java interface method

A prior binding procedure wires the marshaller and the API implementations in the deployment phase. It is important to mention that in a productive environment, multiple web-interfaces may exist which are bound differently to the APIs to allow control for different user roles. For example, normal users are allowed to call a public web-interface that binds the IQueryAPI and IUserConfigAPI only, i.e., all other APIs cannot be requested. In contrast, there is admin web-interface in place that utilizes all APIs but is specially secured for administrative purposes.

In addition to the web-interfaces, the IAdminAPI and IUpdateAPI can be utilized also via JMS which can be used to exchange messages asynchronously as shown later.
8.2.2.2 COMPONENT: RULEENGINE

The RuleEngine component is responsible for the management and execution of complex event processing and business rules based on the gamification domain entities. Figure 8.12 shows the component’s internal structure which represents the implementation of the EPA conceptualized in Chapter 7.

Core component is the complex event processor and business rule engine which implements efficient event processing based on RETE and CEP. For this purpose, Drools Experts and Drools Fusion [70] have been reused and wrapped behind the IRuleEngine interface. Incoming events are processed and delegated by the EventManager which implements the proposed event life-cycle as introduced in Chapter 7 such as point-in-time semantics or eager player checking.

Based on the incoming events the RuleEngine uses the JavaSDK component to retrieve state from the domain entities utilizing the interfaces defined by the AchievementProvider. Depending on the component configuration, the JavaSDK either calls synchronously to the AchievementProvider or provides status over its local in-memory storage. Correspondingly, the JavaSDK updates status synchronously or asynchronously with the AchievementProvider.

Prior to deployment of rules in the CEP component, the RuleService manages the life-cycle of rules, i.e., creations, updates, and deployments. Again, this component requires the AbstractAPI as common implementation for all APIs. The dependencies to subsequent components equal the descriptions for the AchievementProvider.

The RuleEngine’s APIs, i.e., IRuleService and IEventManager, are exposed as web-interfaces. Alternatively, the latter is also accessible over JMS to support asynchronous messaging scenarios.
8.2.2.3 COMPONENT: GATEWAY

In general, the Gateway can be considered as a proxy for the AchievementProvider and the RuleEngine as it encapsulates both components transparently for the client.

In its simplest form, this component exposes the interfaces of the already introduced JavaSDK as web interfaces. Based on the configuration, the components are either accessed directly using JsonRPC calls or indirectly over JMS.

In addition to this simple delegation mechanism, the APIs are exposed as JsonRPC, plain RESTful service, or SOAP-webservice (Simple Object Access Protocol). Furthermore, all requests are checked with regards to standard security mechanisms such as Single Sign-On (SSO), Cross-Site-Request-Forgery (XSRF), or Cross-Site-Scripting (XSS). If configured, enhanced security mechanisms can be utilized such as user anonymization. Some of these mechanisms (e.g., XSRF or anonymization) are implemented by the SecurityPackage component, while others (e.g., SSO or user management) are consumed via JNDI services.

Finally, this component might be also used as load balancer or monitor for high-availability scenarios. In the first case, requests can be dispatched to different AchievementProviders or RuleEngines when utilization of resources becomes too high. In the latter case, the gateway may mask failures of the downstream components. A precise definition of these additional features is not explained further. Instead, Chapter 9 gives an outlook on additional engineering aspects which are not covered in this evaluation.

8.2.2.4 COMPONENT: ADMINUI

To manage and maintain the content within the gamification platform, there is an administration UI in place which is presented exemplarily in Figure 8.14. This user inter-
face comprises several management views to control the runtime system, especially it provides a graphical interface to all APIs of the back-end components. Regulated by the security services, only administrators can access this panel.

Figure 8.14 shows on top the overview page that presents the cardinalities of current game mechanics at a glance. In the lower picture, the rule editor for writing and editing rules is shown as example\(^5\).

**8.2.2.5 COMPONENT: GAMIFICATION WIDGETS**

Beside the AdminUI, there are several independent gamification front-end widgets available that support the integration of visual gamification aspects into the host application. Therefore, this component consists of platform-dependent artifacts, e.g., for web or mobile applications where the gamification platform is later integrated. These widgets have several advantages for developers of the target application. First, they allow for rapid introduction of gamification into the target application as they hide the complexity of communication with the back-end and reduce the amount of code that

\(^5\)Supplementary pictures can be found in Appendix D.
Evaluation

has to be implemented repeatedly in every application. Second, updates to the widgets can be introduced on the fly without touching the host application. For example, the look and feel of widgets can be configured on the server side or new widget features can be delivered without the necessity of application recompilation and redistribution. Third, in some embodiments such as web applications, developers do not need to care for security mechanisms such as Same-Origin-Policy (SOP) and its implications. For example, in the case of SOP, a developer who writes his own widgets has to either place the widgets on the gamification server\(^6\) or has to reimplement the system’s provided web interfaces within the host application delegating all calls to the actual gamification runtime environment in the background.

Each widget is equipped with a small API that also allows for client-specific configurations. Some of these settings can be either defined on the client or server side. If one particular option is set on both sides, then the server-specified configuration is the dominating one.

The following text describes some of the available widgets and their use. Notifications can be, for example, integrated on web-sites using pre-implemented scripts. Listing 8.2 shows how the host application may consume the notification script from the running platform instance. The client then receives notifications accordingly with no additional effort as shown in Figure 8.15. Additionally, this script allows for client-side configuration such as setting the user’s name (i.e., for whom notifications are retrieved), the style (i.e., how the widget looks graphically), or if notifications for entities should be displayed or not.

\[
\begin{align*}
1 \quad & <\text{script src}="https://domain.com/gamification/sap_gp_notifications.js" \text{id}\ "SGPNotificationScript"> \\
2 \quad & SGPNotifications.setUserName("some.player@domain.com"); \\
3 \quad & SGPNotifications.addStyleClasses(["customStyle"]); \\
4 \quad & SGPNotifications.showPoints(true); \\
5 \quad & </\text{script}>
\end{align*}
\]

Listing 8.2: Programmatic integration of notification widgets

As another example, Figure 8.16 shows the usage of a mission widget which requires the user to accept, reject, or temporarily ignore an offered goal. The mission widget hides all communication that is necessary with the back-end such as checking for new missions and sending accept or reject events. Listing 8.3 shows the integration code. The widget script is dynamically retrieved from the corresponding back-end.

\(^6\)This is considered as not practicable when the gamification platform is consumed as an external managed service.
8.2 Implementation

Figure 8.17: Assemble of different gamification widgets

The developer may configure the widget, e.g., by setting the user name, root mission of the corresponding storyline, and the ID of the container where the widget should be bound to.

```javascript
<script src="https://domain.com/gamification/sap_gp_missions.js">
var config = {
  username : <userid>,
  rootMission : <missionID>,
  showIn : <divID>,
  colorPrimary : #550000
  colorSecondary : #FFFFFF
}
SGPMissions.init(config);
</script>
```

Listing 8.3: Programmatic integration of mission widget

Figure 8.17 shows additional submodules such as profile widgets containing the avatar, a mission widget showing the user’s current progress, or a badge showcase. All of these widgets can be consumed in a similar fashion as presented for the notification and the user-accepted mission widget.

Besides widgets for the web-sites, this component consists of further modules for other target platforms. For example, Figure 8.18 shows the integration of widgets on the iOS platform integrated into the Financial Fact Sheet application (Section 8.1.3).

The integration effort is comparable to web applications. First, the mobile framework has to be configured and initialized as shown in Listing 8.4.

```objective-c
[[GamificationDataCenter defaultCenter] initPlatformWithURL:@"<https://targetgamificationplatform.com>>" useSSO:YES];
```

Listing 8.4: Initialization of mobile gamification widgets

Then, a profile or mission view might be allocated and added to one of the existing views of the host application as shown in Figure 8.5 and 8.6.

```objective-c
PrivateProfile *playersProfile = [[PrivateProfile alloc] initWithFrame:self.view.bounds];
```
8.2.2.6 SYNCHRONOUS DEPLOYMENT MODE

So far, the individual components with their internal structure and externally provided and required interfaces have been shown. The assembly of these individual components reflecting the overall implemented infrastructure for the synchronous mode is presented in Figure 8.19.

For the sake of a clear presentation, Figure 8.19 presents all server side components on the same logical web server, i.e., the Gateway, AchievementProvider, and RuleEngine all reside on the same node. Internally the Gateway communicates over the web interfaces of the AchievementProvider as well as the RuleEngine (i.e., via loopback IP address). Also the required interface of the RuleEngine is wired internally to the provided web interface of the AchievementProvider.

7Since all components can communicate over RPC, they might be of course distributed across multiple physical nodes in practical embodiments.
8.2 Implementation

Figure 8.19: Component diagram of gamification platform in synchronous mode

In the deployment phase of the web bundles, both components are also dynamically bound to Java Database Connectivity (JDBC) interfaces provided by the infrastructure's runtime environment. Hence, both components may communicate with a common database using the JPAManger component. The same applies to the SecurityServices which are bound dynamically at deployment, e.g., the gamification platform is registered as service provider at the identity provider to allow for SSO.

The AdminUI is an external component that utilizes the Gateway's APIs to access the back-end. In a practical scenario, the AdminUI is static web content which is delivered to the client and resides there as an instance (e.g., in the client's browser). Therefore, Figure 8.19 shows an instance of the AdminUI component residing on the client side.

Finally, the gamified host application communicates with the gamification platform over the web interfaces either directly using the web interfaces of the Gateway or indirectly using the provided Gamification Widgets. In this thesis not all possible integration scenarios are considered. Nonetheless, the typical and proposed scenario is that the back-end of the host application is instrumented to send relevant user actions and other events to the gamification platform. On the front-end side, widgets from the Gamification Widgets component might be reused to enable the visual aspects and appearance of gamification elements.

8.2.2.7 ASYNCHRONOUS DEPLOYMENT MODE

Figure 8.20 shows the component orchestration when the platform is configured in asynchronous mode. For the sake of simplicity, the same deployment structure as in the synchronous mode is used.

In contrast to the synchronous mode, three message channels separated by topic (marked with the UML stereotype <<topic>>) are established over a message broker (ActiveMQ) running in publish/subscribe mode. As UML does not explicitly support the notion of message queues, each provided-required pair annotated with <<topic>> can be considered either as observer pattern, i.e., the receiver subscribes to a topic of interest and subsequently receives all published messages for that topic or as an endpoint for publishing messages, i.e., the sender publishes tuples for the topic. In the first case (subscription), the corresponding arc is annotated with “receive”, in the latter case with “publish”.

The three utilized topics are UserActions, Consequences, and Notifications. The UserActions topic is used to publish all events from the host application. This topic
has the RuleEngine as the only subscriber which receives all events issued by the host application.

Over the Consequences topic all events created by the RuleEngine component are published, i.e., when gamification relevant messages have to be sent. For this topic, the AchievementProvider is the only subscriber, thus, receiving all updates from the RuleEngine.

The message loop for the client is closed by the third Notifications topic. Here, the RuleEngine and the AchievementProvider are publishing messages. The subscribers for notifications are the host application or the Gamification Widgets embedded within.

Since only the IUpdateAPI and IAdminAPI are provided over the JMS channel, there are still standard web interfaces available which might be utilized by the host application to retrieve entity state when required.

This description concludes the implementation section for the runtime environment. In the next section, it is described how this platform has been utilized to implement and run various applications as introduced in Section 8.1.

8.3 INTEGRATION

In this section, the actual integration scenarios for two selected applications are shown. First, this includes a representative excerpt from the GaML instance describing the gamification concept. Second, the integrated architecture comprising the host application and gamification runtime environment is presented.

8.3.1 SAP TWOGO

Listing 8.7 shows parts of the SAP TwoGo gamification concept described in Section 8.1.1.

In this concept two user actions, namely a rideIntent and ride, are defined. The first declares a meeting request ID (mrid) property as well as its inverse action deleteRideIntent that joins on the request ID. The latter, additionally, signifies through the carbon and the kilometers which were saved in the ride, if the user who issued the event was the driver, and an ID that uniquely identifies the number of the ride.

Furthermore, there is an internal event called socializerEvent which is used to represent all users who shared a ride with each other.
The concept features four different point categories, namely Socializer, Experience, Kilometer, and Carbon points. Based on the Experience point category several levels such as Rookie or TricycleRider are defined.

```ruby
concept CARPOOL {

useraction rideIntent {
  properties { mrid:Decimal }
  inverseEvent { name = deleteRideIntent, joins=mrid }
}

useraction ride {
  properties { carbonImpact:Decimal, kilometers:Decimal, driver:Boolean, rideId:Number }
}

internaevent socializerEvent { properties { pid:Text, friend:Text } }

point Socializer { name="Socializer Point", abbreviation="SP"
  when player { did evt1 : useraction ride and another player did evt2 : useraction ride, rideId=evt1.rideId and not did internaevent socializerEvent, pid=evt1.player, friend=evt2.player } then {
    create event socializerEvent(pid=evt1.player, friend=evt2.player)
  }
  when player { did internaevent socializerEvent } then { give 1 Socializer }
}

point XP { name="Experience Point", abbreviation="XP", type=ADVANCING
  when player { did useraction rideIntent, lastsFor 24h } then { give 1 XP }
  when player { did evt1:useraction ride, driver=false } then { give 10+evt1.kilometers/10 XP }
}

level Rookie { name = "TwoGo Rookie", threshold = 5 XP }
level TricycleRider { name = "Rider of a Tricycle", threshold = 10 XP }
//...

point Carbon { name="Carbon Point", abbreviation="CP"
  when player { did evt1:useraction ride }
  then { give evt1.carbonImpact/10 }
}

point KilometerPoints { name="KM Points", abbreviation="KMP"
```
The Experience metric also defines two direct rules how points for this category might be achieved. First, users may receive one XP when they are creating a ride intent to signify their willingness to share a ride. This intent has to reside at least for 24 hours in the system, i.e., there is no inverse user action deleteRideIntent in between. If so, the user receives one XP. Furthermore, for each actual ride users receive one-tenth of their saved kilometers plus ten as Experience points.

Similarly, kilometer and carbon points are given when a ride has been done. In particular one-tenth of the saved carbon is added as points as well as 1.6-times the saved kilometers as kilometer points.

Finally, Socializer points are given when a socializer event occurs. However, this event cannot be issued by users directly but is generated when the following situation happens: if users are doing a ride together, all of them are sending their ride event through the application to the gamification runtime system. The first condition in the Socializer point definition says that if two different players did a ride with each other and never had a ride before (i.e., there is already a socializer event available), then a socializer event is created (Listing 8.7, line 16-22). In this case, the second rule applies and gives the player the point (Listing 8.7, line 24f.).

This basic concept has been compiled into executable code deployed within the gamification runtime system. In order to receive events and calculate the game mechanics, the runtime was integrated with the actual host application, i.e., the TwoGo system. Figure 8.21 shows the integration architecture. Please note, that in the following the gamification runtime environment is considered as one large component called Gamification Platform with no further assumption on how the components described in Section 8.2.2 are structured.

The TwoGo system itself schematically consists of a Microsoft Outlook plug-in that manages incoming calendar meeting requests signifying ride intents, a Matcher that matches users based on their intents, and a front-end for the user interaction. The host application uses the JavaSDK component introduced in Section 8.2.2 to communicate with the gamification platform in a transparent way. In particular, the Outlook plug-in, and the Matcher are using the IEventManager interface to communicate ride intents or rides respectively. Furthermore, the TwoGo front-end required gamification data over the provided web interface using the provided IQueryAPI. This applies also to the Matcher which may use current gamification data to influence the matching algorithm.

In this embodiment, the provided gamification widget component was not used. Hence, the front-end implements its own graphical elements to represent the gamification as shown previously in Figure 8.1.

### 8.3.2 SAP NETWORKING LUNCH

An excerpt of the gamification concept expressed in GaML for the Networking Lunch application is given in Listing 8.8.

Altogether, there are seven explicit user actions and one internal event which are used to track the corresponding scores. In fact, for each of the user actions, there is
one corresponding point category which is, usually, increased by one when the event happens. For example, the `addBuddy` event increases Buddy points. Only the `AcceptedMeeting` category is computed slightly differently as it is reset to zero if a `declineMeeting` user action occurs.

Similar to the TwoGo case, combinations of two users from different cost centers who attended a meeting in real-life are joined together signified through an internal `meeting` event.

```java
concept NetworkingLunch {
    useraction attendMeeting {
        properties { costCenter: Number, _type: Text }
    }

    internalevent meeting {
        properties { pid1: Text, pid2: Text, costCenter1: Number, costCenter2: Number }
    }

    useraction addBuddy
    useraction addTag
    useraction addNote
    useraction addedAvailability
    useraction acceptedMeeting
    useraction declineMeeting

    point MeetingPoints {
        name="Meeting Points",
        abbreviation = "MP"
        when player {
```
did evt1 : useraction attendMeeting
and another player did evt2 : useraction attendMeeting, costCenter<>
evt1.costCenter
and not did internalevent meeting, pid1=evt1.player, pid2=evt2.player
} then {
create event meeting(pid1=evt1.player, pid2=evt2.player, costCenter1=
evt1.costCenter, costCenter2=evt2.costCenter)
}

when player {did internalevent meeting }
then { give 1 }
}

point AcceptedMeetings {
    name = "AcceptedMeetings"
    when player {did useraction acceptedMeeting}
    then { give 1}
    when player {did useraction declineMeeting}
    then { set 0 }
}

point Attended1_1Lunch {
    name="Attended1:1Lunch"
    when player { did useraction attendMeeting, _type="1:1" } then { give 1 }
}

point Buddies {
    name = "Buddies"
    when player { did useraction addBuddy } then {give 1 }
}

point Tags {
    name = "Tags"
    when player { did useraction addTag } then { give 1 }
}

//...
badge ReadyToNetwork {name="ReadyToNetwork", image="readytonetwork.png"}
badge OnTime {name = "OnTime", image="ontime.png"}
//...
badge FullHouse {name = "FullHouse", image="fullhouse.png", hidden}

mission startmission1 {
    name="Get Ready to Network",
    description="Get Ready to Network",
    initiatedBy=rule
    when player {
    has point Availabilities, SUM >= 1
    and has point Tags, SUM >=1
} then {
    give badge ReadyToNetwork
}
}

mission startmission2 {
    name = "On my Calendar",
Based on the point categories, there are several missions modeled whereas the missions `startmission1`, `startmission2`, and `startmission3` are defined as entry missions as they do not define an availability clause. In addition, these missions are declared to be rule-driven, i.e., they are assigned automatically to each user that enters the system. To fulfill, for example, the mission `startmission1`, users have to create at least one availability time slot and one tag expressing their preferred conversation topic in the host application. Afterwards, the `ReadyToNetwork` badge is given. Further, the mission `PeoplePerson` comprises a pre-condition which requires either the second or the third mission to be completed. If this pre-condition is met, the mission is assigned to the user. For the sake of brevity, only exemplary rules are shown, i.e., this case defines further 25 missions in a similar way.

The remaining missions, typically, check if one of the point categories exceeds a particular threshold, gives one or more badges, and assigns the next missions. For example, the `PeoplePerson` mission checks if the user has accepted 25 meetings in a row...
and gives the *FullHouse* badge. Furthermore, the completion leads to the fulfillment of the *Accept25Meetings* mission’s pre-condition etc.

![Integration architecture for SAP Networking Lunch application](image)

Figure 8.22: Integration architecture for SAP Networking Lunch application

From a technical perspective, Figure 8.22 shows the integration of the gamification into the host application. Very similar to the TwoGo case, there are a front-end, a core application, and a matcher. In contract, the back-end uses the *JavaSDK* component only to issue events against the gamification platform’s *IEventManager* interface. State, on the other hand, is retrieved by the gamification widgets which have been weaved directly into the front-end of the Networking Lunch application. As described before, these widgets use the web-interface (primarily the *IQueryAPI* interface) to render the gamification state of the player.

### 8.4 PERFORMANCE ANALYSIS

In this section, the overall runtime behavior and performance of the proposed gamification environment is investigated in two steps.

First, the Networking Lunch application (Section 8.1.2) is used as an example to investigate the overall performance and compare the different deployment options with each other. Furthermore, the measurements are compared against a pure CEP implementation which has very high performance but lacks persistent state data. This measurement is used as performance baseline to compare all other proposed solutions against it.

Second, the statements of the performance behavior are generalized by investigating the impact of individual factors towards performance. These factors are then combined into a general performance model for the deployment modes. It is shown that this model enables the forecasting of average response times and throughputs with an acceptable estimation error.
8.4 Performance Analysis

8.4.1 EVALUATION SETUP

For the first part of the performance evaluation, the Networking Lunch application is utilized as it includes a representative set of rules, i.e., it contains rules that heavily require or update state upon rule processing. Overall the evaluation is based on 46 complex event processing and production rules. Given the types of rules from Section 7.3, Table 8.1 shows the quantities per rule type, i.e., the rule set contains 30 rules that require context in each evaluation cycle within its LHS and updating the context in the RHS. Further 15 rules take events in the LHS and update context on the RHS.

Table 8.1: Quantities of rule types in the SAP Networking Lunch application

<table>
<thead>
<tr>
<th>RHS</th>
<th>LHS (a)-(d) (e)-(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(g)</td>
<td>1 0</td>
</tr>
<tr>
<td>(h)-(i)</td>
<td>15 30</td>
</tr>
</tbody>
</table>

The experiments are performed for $2^n$ experimental users with $n = 3, \ldots, 12$. Each user creates on average 0.67 events/second which is extrapolated based on the current usage statistics of the application, whereby user behavior was extrapolated by replacing day with seconds at constant event probabilities to test in fast motion and to avoid the overhead of too many simulation threads. Overall, one simulated user corresponds to

$$\frac{0.67 \text{ Events}}{s} = \frac{0.67 \text{ Events} \times 86400 \text{ s}}{0.67 \text{ Events} \times s} = 86400 \text{ Users}. \quad (8.1)$$

Given the assumption that the user behavior can be linearly extrapolated, the system’s behavior for $2^n$ experimental users corresponds to the behavior $2^n \times 86400$ Users, i.e., $6.91 \times 10^5$, $1.38 \times 10^6$, $3.54 \times 10^8$ “real users”.

Eight different event types or user actions (e.g., addBuddy, acceptMeeting, or addTag (Section 8.3.2)) are created for all possible actions users can perform in the application. For the experiment, each event type is triggered with a certain probability defined by the application scenario. These probabilities are fixed for each user, and thus, in an observation time of 5 minutes, the average number of events in total is

$$2^n \text{ Users} \times 0.67 \text{ Events} \times \frac{1}{\text{s}} \times 300 \text{ s} = 201 \times 2^n \text{ Events}. \quad (8.2)$$

In order to take measurements, the codebase has been instrumented with measurement points on which loggers passively write the currently processed data tuple including timestamps. Timestamps are recorded with System.nanoTime() which provides a higher resolution (< 10µs) than System.currentTimeMillis() (15ms) [97].

Figure 8.23 shows the locations of measurement points for the synchronous deployment model. The presented queuing network model consists of the core components which are involved in the overall processing shown as nodes. Two queues are involved at the CEP and BEP nodes representing the request queues of the server. These queues schedule incoming request with the first-come, first-served (FCFS) strategy.
At each component (node) the entry time $t_{in}$ and exit time $t_{out}$ of the processed tuple are measured. All measured entry and exit times are stored in the sets $\mathcal{T}_{in,j} = \{t_{in,1,j}, t_{in,2,j}, ..., t_{in,k,j}\}$ and $\mathcal{T}_{out,j} = \{t_{out,1,j}, t_{out,2,j}, ..., t_{out,k,j}\}$ respectively with $j$ representing the $j$-th iteration starting at the entry point of the CEP node and ending at the exit point of the BEP node. In each iteration, a tuple may pass the CEP node multiple times based on the deployed rules. Hence, each iteration has its own number of passed services expressed through $k_j$. The processing or service time at the $i$-th node in the $j$-th iteration is $t_{p,i,j} = t_{out,i,j} - t_{in,i,j}$ with $i = 1, ..., k_j$. Hence, the total service time $t_{st}$ for one tuple is

$$t_{st,j} = \sum_{i=1}^{k_j} t_{p,i,j}$$  \hspace{1cm} (8.3)

excluding waiting times in queues. Consequently, the overall processing time $t_a$ for one iteration is $t_{a,j} = \text{max}(\mathcal{T}_{out,j}) - \text{min}(\mathcal{T}_{in,j})$, i.e., including service and queuing times. For the following text, the time $t_{a,j}$ is considered as the general response time$^8$. In the following experiments, the discrete distribution of $t_a$ over all iterations and all user requests for the selected Networking Lunch application is measured. Based on this distribution, standard scores such as average or percentiles are presented.

It is important to note that subsequent rules might be triggered upon state update in the BEP. Since $t_a$ considers only the unidirectional passing of nodes including loops at the CEP node, the overall processing time $t_{total}$, i.e., the time until the entire system is in a consistent state after the arrival of one event, is:

$$t_{total} = \text{max}(\mathcal{T}_{out,i}) - \text{min}(\mathcal{T}_{in,1})$$  \hspace{1cm} (8.4)

with $l$ being the number of iterations.

Correspondingly, Figure 8.24 shows the measurement model for the asynchronous mode. Additional measurement points were introduced at the message broker, local BEP, and JavaSDK that holds the state context.

As the underlying technology platform is based on the Java programming language, reliable performance results are difficult to achieve as many different factors influence its accuracy. Even very simple Java classes (e.g., 32-bit linear feedback shift register) lead to deviations in performance results [33]. Factors for unreliable measurements include cache misses, multi-threading and the synchronization of threads, or memory

---

$^8$Please note that the general response time $t_{a,j}$ does not include delay times in the CEP node caused by time windows or temporal rules as intended or imposed by the gamification design. For example, an intended time delay might be caused when joining two confirm actions of two different users. Since the second event may arrive with a time delay, the response for the first event cannot be computed immediately but only upon arrival of the second event only. Since the selected application scenario does not contain such constraints, those delay times are avoided and not considered for the sake of simplicity.
management functionalities such as garbage collection or object finalization in the Java-Virtual Machine (VM) [33, 34]. Furthermore, the operating system may foster context-switches of the Java-VM process among processes or processing units when multiple physical processors are present. In addition, the Java-VM uses various techniques to optimize the code at runtime such as Just-in-time (JIT) compilation or lazy class loading. In the first case, Java programs run system independently using bytecode. This bytecode is interpreted at first. Meanwhile, the Java-VM collects statistics, e.g., number of method invocations and branches. Based on these statistics, JIT compilation is used to determine and compile parts of the bytecode into optimized machine code for the underlying system after a particular threshold of execution steps for the considered code part is exceeded. This results in faster running code. Since this process is completely transparent to the developer, consequently, there is no guarantee when optimization of the code is finished. In order to get reliable performance measurements, this so-called warm-up phase has to be excluded from the collected data.

Considering these specifics, the experiments were run under the following conditions. First, all components (one web server with all web applications deployed) run within the same single VM process and load was produced from a separate Java-VM process. Second, the operating system was configured to run each Java-VM on a separate physical processing core (with hyper-threading disabled) and all other non-Java-VM processes were scheduled on the remaining physical cores using CPU affinity flags. Third, memory-specific settings such as heap size (-Xmx, -Xms) and the size of permanent generations (-XX:MaxPermSize) were set to the maximum hardware resource of the machine in order to avoid unnecessary garbage collection cycles by the Java-VM. Fourth, after each experiment, the Java-VM was forced multiple times to do garbage collection and object finalization, since the respective functions are only a hint for the Java-VM and, thus, do not provide guarantee that all stale objects have been removed from the heap.

The specification of the underlying hardware is a machine with 32GB RAM, 16-cores Intel Xeon L5640 processors, and Windows Server operating system.

### 8.4.2 EXPERIMENTAL RESULTS

Besides the synchronous and asynchronous strategies from Section 7.3, the following text refers to two additional strategies that are included for comparison reasons.

First, a plain CEP implementation is used as a baseline measurement. Based on prior research it is assumed that this implementation has the highest performance
with regards to processing of events and facts in the main memory. Therefore, in this variant, there is no explicit AchievementProvider that persists state but all events and facts are kept in the rule engine’s volatile working memory as dictated by the general approach.

Second, for the evaluation an additional strategy as mix of the described synchronous and asynchronous approaches is included. Here, the implementation and the data of the query interface resides locally at the EPA, hence, allows for fast local state retrieval. Updates, on the other hand, are synchronized with the local state repository and synchronously updated with the remote BEP. Hence, this strategy behaves basically like the asynchronous approach except that updates to the BEP are replicated synchronously. Although this case would occur rarely in practice, it is included for theoretical consideration since it combines the proposed synchronous and asynchronous strategies.

Figures 8.25a and 8.26 present the average and maximum response times for all four approaches.

In the average case, the asynchronous strategy excels the synchronous strategies significantly up to seven orders of magnitude (1024 users). It is important to note that beyond 1024 experimental users, the synchronous case is not only significantly slower, but due to limited queues, the server starts to reject requests after a particular timeout which results in errors and makes results incomparable. Therefore, synchronous data is only presented up to 1024 users. For the asynchronous case, event rates beyond 4096 users reached the capacities of the simulation environment. Before this point, i.e., until 2048 experimental users, the proposed asynchronous approach is only slightly slower compared to the plain CEP baseline measurement (i.e., 0.98ms-4.31ms).

In the maximum case, the difference between the synchronous and asynchronous approach is smaller (four orders of magnitude for 1024 users). However, analyzing the underlying response time distributions yields that large response times are much more likely in the synchronous case, i.e., 95% of all responses are slower than $10^6$ ms. The

![Figure 8.25: Experimental average response times (in ms)](image)
distribution of the asynchronous case shows that only 1% of the responses are slower than $10^5$ ms but 98% are faster than 100 ms.

(b) Response Time Max

Figure 8.26: Experimental maximum response times (in ms)

For example, Figure 8.27 shows the response time distributions at 1024 experimental users for the synchronous (a) and the asynchronous (b) case respectively. While the synchronous approach has to deal frequently with high response times, the asynchronous case only shows such high response times in rare cases.

(a) 1024 Users
(b) 1024 Users

Figure 8.27: Comparison of response time distributions between synchronous and asynchronous case at 1024 experimental users

Furthermore, two phases can be distinguished in the average case. First, the phase in which the systems runs normally. In this phase, the data distribution is highly right-skewed as shown in Figure 8.28a. Second, the phase where the system runs under high load. This phase is characterized through an increasing frequency of high response times (e.g., Fig. 8.28b - 8.28d). For example, in the synchronous case, the first phase persists until 16 users. The second phase starts at 32 users and ends at 1024 users,
i.e., when tuples are rejected from the waiting queues. It is important to note that Figure 8.28d corresponds to Figure 8.27 but with a different scaling on the y-axis.

In the asynchronous case, the first phase persists until 2048 users, i.e., two orders of magnitude later with regards to the amount of events (Eq. 8.2).

![Figure 8.28](image)

Figure 8.28: Evolution of response times for synchronous communication

This statement is further supported by Figure 8.29, where a response time threshold of 500ms is considered. Already for 8 users, 8% of all events are too late in the synchronous case. For 256 users, already 95% of the events are above the threshold. In contrast, events are delayed in the asynchronous case starting at 512 experimental users (3%).

Nevertheless, the speed-up in the asynchronous case comes at the cost of growing memory in the EPA. The resulting space-time trade-off is presented in Figure 8.30 where the speed-up is compared to the inverse memory consumption of the asynchronous approach. As presented, the intersection can be found around 32 experimental users, i.e., for small workloads the synchronous case might be preferred. Larger workloads may be deployed in an asynchronous setting. This trade-off, however, does not take into account the individual costs for performance or memory. For example, at the presented trade-off point already 60% of events are highly delayed in the synchronous case which might not be acceptable either. Thus, a cost-benefit trade-off might look different. Overall, the compromise of memory and response time is not generalizable but depends on the workload, rules, and deployment model of the experimental setup described above. Because of these limitations, Section 8.4.4 develops
8.4 Performance Analysis

**Figure 8.29:** Slow response times compared to a fixed 500ms threshold

**Figure 8.30:** Space-time trade-off of response-time versus inverse memory consumption in the event processing agent

A generic performance model that can be used to forecast the runtime behavior of the gamification system for arbitrary gamified application scenarios.
8 Evaluation

8.4.3 DISCUSSION

As shown above, the plain synchronous system is up to seven orders of magnitude slower and runs already under overload at 32 experimental users. Moreover, data tuples are discarded under overload conditions leading even to wrong results in the BEP. Holding the context local in the EPA with synchronous communication shifts the overload phase to 512 experimental users. Therefore, the synchronous communication may not acceptable in cases where the system has to process high rates of events.

The asynchronous case with local BEP is able to compete with the baseline CEP measurement in the median or average case. The overload phase occurs at 4069 users. The same holds for the pure CEP implementation. Hence, it is argued that the hybrid concept of Section 7.3 with asynchronous communication serves all the requirements of Section 4.3 including high performance in the asynchronous mode.

8.4.4 GENERIC PERFORMANCE MODEL

So far, the performance results provide a strong indicator for the runtime behavior of the overall system. However, the results are not generalizable as they can be interpreted only in the context of the considered application, namely the Networking Lunch application. The main reason for this specificity is that the previous consideration is a naïve regression of the number of users on the average response time \( t_a \) in a given application, i.e., omitting the parameters and resulting behaviors of the individual components within the presented performance models (Figures 8.23 and 8.24). For instance, the performance behavior of the CEP node in Figure 8.23 depends on the number and characteristics of the deployed production and event processing rules. Based on these parameters, the node's response time may increase or decrease or the number of output tuples changes which affects the behavior of subsequent nodes. The modeling of these aspects is described in the following text.

As part of this thesis, [40] investigated the different performance behaviors \( t_p \) for each node of the performance models individually and provides a more general model for the runtime environment to model and forecast the average response time \( t_a \). In fact, queuing networks are used to simulate and forecast \( t_a \) based on the statistical dependencies between single services with well-known performance behavior using response time functions of \( t_p \). The resulting model might be used then by practitioners to forecast the performance behavior of the system and to decide for the correct deployment and infrastructure options.

As preliminary consideration, 20 factors that impact the overall system performance were identified in a quantitative pre-study. Table 8.2 shows which factors have been studied and found twelve relevant factors for the system’s overall performance.

Based on the pre-study non-relevant and some of the relevant factors were excluded from further consideration as they only had a minor impact on performance. The set of remaining factors \( \mathcal{F} = \{F_1, F_2, ..., F_8\} \) is marked as modeled in Table 8.2.

For each considered factor, all services or nodes \( S_{\text{mode}} \) of the performance models (Section 8.4.1) for the synchronous (Figure 8.23) and asynchronous (Figure 8.24) modes were measured individually. Hence, there is exactly one response time function \( t \) for each combination of service \( s_i \), factor \( F_j \), and the selected mode \( \text{mode} \in \{\text{sync, async}\} \):

\[
t_{s_i,F_j,\text{mode}} : \mathbb{R}_+ \rightarrow \mathbb{R}_+ \tag{8.5}
\]
Table 8.2: Summary of examined cost factors in the performance model: X = yes; – = no; ? = unknown [40]

<table>
<thead>
<tr>
<th>Cost Factor (#)</th>
<th>Relevant Modeled</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Users</strong></td>
<td></td>
</tr>
<tr>
<td>Number of Users (1)</td>
<td>X</td>
</tr>
<tr>
<td>Events/s/User (2)</td>
<td>X</td>
</tr>
<tr>
<td>Event Type Distribution (3)</td>
<td>X</td>
</tr>
<tr>
<td><strong>Rules</strong></td>
<td></td>
</tr>
<tr>
<td>Number of Rules (4)</td>
<td>–</td>
</tr>
<tr>
<td>Number of Alpha Nodes (5)</td>
<td>X</td>
</tr>
<tr>
<td>Number of Beta Nodes (6)</td>
<td>X</td>
</tr>
<tr>
<td>Node Types (7)</td>
<td>X</td>
</tr>
<tr>
<td>Number of Abstractions (8)</td>
<td>X</td>
</tr>
<tr>
<td>Working Memory Growth Rate (9)</td>
<td>–</td>
</tr>
<tr>
<td>Independent Rule Streams (10)</td>
<td>?</td>
</tr>
<tr>
<td>Number of UpdateAPI calls (11)</td>
<td>X</td>
</tr>
<tr>
<td>Number of QueryAPI calls (12)</td>
<td>X</td>
</tr>
<tr>
<td>Size of Tables (13)</td>
<td>–</td>
</tr>
<tr>
<td>Structure of Tables (14)</td>
<td>?</td>
</tr>
<tr>
<td><strong>Infrastructure</strong></td>
<td></td>
</tr>
<tr>
<td>Database (15)</td>
<td>X</td>
</tr>
<tr>
<td>Queue Scheduling Algorithm (16)</td>
<td>–</td>
</tr>
<tr>
<td>Connection Pools: max. Number of Threads (17)</td>
<td>–</td>
</tr>
<tr>
<td>Connection Pools: Accept Count (18)</td>
<td>–</td>
</tr>
<tr>
<td>Transmission Packet Sizes (19)</td>
<td>?</td>
</tr>
<tr>
<td>RAM of Server (20)</td>
<td>–</td>
</tr>
</tbody>
</table>

with $s_i \in S_{mode}$ being $i$-th service in the set of all services for the selected mode and $F_j \in F$ the $j$-th performance factor. Each $F_j$ is again a set of values characterized by the factor’s nature (e.g., $F_{Number \ Of \ Users} = \{50, 100, ..., 1000\}$, $F_{Events \ Per \ UserSecond} = \{0.1, 0.15, ..., 1\}$, etc.). For each discrete value $v \in F_j$ the simulation was executed for five minutes, hence, there are samples of $N$ values available which are aggregated by mean:

$$t_{s_i,F_j,mode}(v) = \frac{1}{N} \sum_{k=1}^{N} t_{s_i,F_j,mode,k}(v), \ \forall v \in F_j$$

(8.6)

The simulation of events across all experimental users from the event source always complied with an exponential distribution (which is also consistent with the assumption

\[9\] Recall that $t_{p,F_{mode}}$ is a more precise definition of $t_p$ based on the newly introduced modeling context.
of queuing network models). Figure 8.31, for example, shows the experimental density of the inter-arrival times of events (i.e., the time between two consecutive events) for 250 users issuing 0.5 events per second. This results in a negative exponential distribution with an average inter-arrival time of 8ms.

Afterwards, the collected data was statistically analyzed. Based on the measured and aggregated service times \( t_{s_i,F_j,\text{mode}} \), the Ordinary Least Squares (OLS) method was applied to retrieve polynomial models. In fact, \( M + 1 \) different models were estimated with OLS to select the best model based on quality criteria, i.e.,

\[
\hat{t}_{s_i,F_j,\text{mode}}(v) = \sum_{m=0}^{M} \beta_{m} v^{m}
\]

To select the best model, \( M \) has been successively increased from one to ten to obtain the constant polynomial \( (m = 0) \) and the \( m \)-th polynomial. With each new parameter the polynomial fits better to the data (i.e., residual errors are minimized). However, to avoid over-fitting and to comply with statistical parsimony, information criteria, namely Akaike Information Criterion (AIC) [4], Bayesian Information Criterion (BIC) [110], and Cross-Validation (CV) [115], were used to determine the model that explains variance best in comparison to the number of regression coefficients. The AIC is, hereby, defined as

\[
AIC = -2\log(L_m) + 2p_m
\]

with \( L_m \) being the maximum-likelihood of the estimated parameters and \( p_m \) the number of coefficients of the \( i \)-th estimated model. Hence, the quality of \( m \)-th model is penalized by the number of regression coefficient. The ratio, where AIC is minimal is selected as the best model.
8.4 Performance Analysis

Since AIC tends to select models with too many parameters on large data sets, BIC normalizes the number of parameters by the sample size \( N \) to further penalize non-parsimonious models.

\[
BIC = -2\log(L_m) + 2p_m\log(N)
\]  

(8.9)

In addition, CV is a method that empirically validates test data against the trained model. For this purpose, CV divides the sample into \( k \) subsets of equal size and estimates the \( i \)-th model from the \( k - 1 \) subsets and tests the model with the remaining subset.

\begin{table}[h]
\begin{tabular}{|l|c|c|c|c|}
\hline
Component & Criterion & \( R^2 \) & \( \beta_0 \) & \( \beta_1 \) & \( \beta_2 \) \\
\hline
CEP & CV & .79 & -.72 & .07 & - \\
Proxy & CV & .83 & .02 & -.72*10^{-4} & .28*10^{-6} \\
BEP Query & BIC/CV & .97 & 1.14 & .01 & - \\
BEP Update & All & .26 & 4.92 & -.01 & - \\
\hline
\end{tabular}
\end{table}

Table 8.3: Service time polynomials for number of users factor in synchronous mode (based on [40])

As an example, Table 8.3 shows the estimation for \( \hat{t}_{s_i,F_{Number of Users_{Sync}}} \), \( \forall s_i \in S_{Sync} \), i.e., the response time functions for all nodes in the synchronous model for the factor Number of Users. It is shown that CV or BIC decide for the most parsimonious model. Additionally, all slopes and intercepts are significant against at least a 10% level. For instance, the CEP node follows a linear function, while the proxy is estimated with a second degree polynomial.

Correspondingly, Table 8.4 shows the estimated processing times at all nodes for factor Number of Users in the asynchronous embodiment, i.e., polynomial estimations for

\[
\hat{t}_{s_i,F_{Number of Users_{Async}}} \quad \forall s_i \in S_{Async}.
\]

The complete set of estimated models for all remaining combinations of services and factors can be found in [40].

After their estimation, all polynomials have been validated in the context of the respective queuing network model using a simulation tool for queuing networks, namely JSIMGraph [27]. For this purpose, the average response time \( \bar{T}_a \) for the entire system, i.e., the services altogether, was predicted in three steps and compared to the measured response times of the running system. First, based on the estimated polynomials, the average service times for each node were forecasted. Second, the service times were given as input to each node of the corresponding queuing network. Third, the simulation was run with JSIMGraph until the overall response time has been predicted.

Afterwards, the predicted values were compared to the measured response time of the overall system. Table 8.5 shows the differences of measured and predicted response times for a selection of scenarios such as 50 users, 100 \( \alpha \) or \( \beta \) nodes in the gamification rules, and 500 abstractions within the rule engine.
Evaluation

<table>
<thead>
<tr>
<th>Component</th>
<th>Inf. Cri.</th>
<th>$R^2$</th>
<th>$\beta_0$</th>
<th>$\beta_1$</th>
<th>$\beta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MB1</td>
<td>All</td>
<td>.97</td>
<td>7.70*10^-1</td>
<td>1.48*10^-3</td>
<td>1.18*10^-6</td>
</tr>
<tr>
<td>CEP</td>
<td>All</td>
<td>.97</td>
<td>4.88*10^-1</td>
<td>-2.10*10^-4</td>
<td>1.90*10^-6</td>
</tr>
<tr>
<td>Proxy</td>
<td>AIC/BIC</td>
<td>.81</td>
<td>3.14*10^-2</td>
<td>2.13*10^-7</td>
<td>-</td>
</tr>
<tr>
<td>CEP Query</td>
<td>All</td>
<td>.17</td>
<td>2.07*10^-2</td>
<td>-3.98*10^-6</td>
<td>-</td>
</tr>
<tr>
<td>CEP Update</td>
<td>All</td>
<td>.13</td>
<td>1.65*10^-2</td>
<td>2.61*10^-6</td>
<td>-</td>
</tr>
<tr>
<td>MB2</td>
<td>All</td>
<td>.04</td>
<td>2.05*10^-1</td>
<td>1.82*10^-5</td>
<td>-</td>
</tr>
<tr>
<td>BEP Update</td>
<td>All</td>
<td>.34</td>
<td>0.56*10^1</td>
<td>6.00*10^-2</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.4: Service time polynomials for number of users factor in asynchronous mode (based on [40])

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Measured</th>
<th>Predicted</th>
<th>MRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 Users</td>
<td>11.46</td>
<td>10.62</td>
<td>.07</td>
</tr>
<tr>
<td>100 Alpha Nodes</td>
<td>402.48</td>
<td>348.07</td>
<td>.13</td>
</tr>
<tr>
<td>100 Beta Nodes</td>
<td>49.56</td>
<td>61.02</td>
<td>.23</td>
</tr>
<tr>
<td>500 Abstractions</td>
<td>343.82</td>
<td>398.90</td>
<td>.16</td>
</tr>
<tr>
<td><strong>Sync</strong></td>
<td></td>
<td></td>
<td><strong>MMRE</strong></td>
</tr>
<tr>
<td>PRED(25)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200 Users</td>
<td>7.55</td>
<td>8.99</td>
<td>.19</td>
</tr>
<tr>
<td>500 Alpha Nodes</td>
<td>22.45</td>
<td>22.13</td>
<td>.01</td>
</tr>
<tr>
<td>100 Beta Nodes</td>
<td>7.71</td>
<td>8.94</td>
<td>.16</td>
</tr>
<tr>
<td>500 Abstractions</td>
<td>223.67</td>
<td>245.30</td>
<td>.09</td>
</tr>
<tr>
<td><strong>Async</strong></td>
<td></td>
<td></td>
<td><strong>MMRE</strong></td>
</tr>
<tr>
<td>PRED(25)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.5: Quantitative validation of cost model comparing measured and predicted response times (based on [40])

As criteria for assessing the prediction quality, Magnitude of Relative Error (MRE) for each scenario as defined by [37] is used:

$$MRE_i = \left| \frac{Actual \ Value_i - Predicted \ Value_i}{Actual \ Value_i} \right|$$ (8.10)

All single MREs are averaged using Mean Magnitude of Relative Error (MMRE):

$$MMRE_i = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{Actual \ Value_i - Predicted \ Value_i}{Actual \ Value_i} \right|$$ (8.11)
or using the Percentage Relative Error Deviation (PRED) measure:

\[
PRED(x) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 
1, & \text{if } MRE_i \leq \frac{x}{100} \\
0, & \text{else}
\end{cases}
\]

(8.12)

as defined by [138, 157]. A good prediction is characterized by a MMRE close to zero or PRED close to 1. Thus, it is argued that individual factors, i.e., where the remaining factors are kept constant, are adequately predicted using the three-step prediction approach as described above.

After the successful prediction and validation of individual, isolated nodes, the next step is concerned with the assessment of the overall performance model. More precisely, in the following step the polynomials are applied to predict the average performance behavior of the Networking Lunch application for 200 users in both, the synchronous and asynchronous, modes.

More specifically, in the synchronous case, these users issued 0.01 events per user-second which equals an average inter-arrival time of 500ms. In the asynchronous case users issued 0.1 events per second which equals an inter-arrival time of 50ms. The RETE graph representing the gamification rules comprised 14 \( \alpha \)-nodes, 8 \( \beta \)-nodes, and 1 abstraction step (Figure D.7). Measurements were taken over five minutes. Furthermore, the rest of the setup as described in Section 8.4.1 applies.

The measurement results for this exemplary case can be found in Table 8.6. Here, the average response time in the synchronous case was 200.26ms and 11.56ms in the asynchronous case. Note that these values refer to the average response time \( \overline{t_a} \) as introduced in Section 8.4.1.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Measured (ms)</th>
<th>Estimation (ms)</th>
<th>MRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 Users / Sync Mode</td>
<td>200.26</td>
<td>287.51</td>
<td>.44</td>
</tr>
<tr>
<td>200 Users / Async Mode</td>
<td>11.56</td>
<td>13.27</td>
<td>.15</td>
</tr>
</tbody>
</table>

Table 8.6: Validation of cost model comparing measured and predicted response times for SAP Networking Lunch application (based on [40])

As well as for the runtime prediction of the single nodes, three basic steps have to be executed, i.e., forecasting concrete services times using the polynomials, inserting the estimated services times into the corresponding nodes of the queuing networks, and simulate the queuing network based on the provided arguments.

The predicted times are shown in Table 8.6 which are taken from the simulation of JSIMGraph as shown in Figure 8.32 for the synchronous case and Figure 8.33 for the asynchronous case. For the former, 287.50ms are forecasted which equals an acceptable MRE of 0.44. For the latter, 13.27ms are forecasted which refers to a good estimation (MRE = 0.15) in the asynchronous case.

Although predictions seem to be adequately based on the proposed performance models, there are many possibilities to improve the estimation quality further. Therefore, the next section discusses the theoretical and practical limitations of the introduced performance model.
8.4.5 DISCUSSION OF GENERIC PERFORMANCE MODEL

Compared to the experimental results of Section 8.4.2, the presented generic performance model provides a better means to estimate the performance behavior of the overall system in both deployment modes given multiple factors which are determined by the gamification concept (i.e., gamification rules), the behavior of the target user in the host application, and additional system factors determined by the underlying infrastructure. In fact, it has been shown that the estimation error of the model is comparatively low (< 100ms in absolute numbers). It is argued that this model provides a suitable means to forecast and judge about the system's behavior in most practical scenarios based on the input parameters before the actual deployment. Thus, it may help practitioners to select the correct deployment mode and infrastructure based on the gamification concept and the host application to ensure perceived real-time feedback.

Nonetheless, this model still possesses drawbacks with regards to generalization and estimation quality. First, some factors which have shown a minor performance
impact were excluded due to the results of the pre-study. However, also these factors influence performance and their omission may affect prediction quality negatively.

Second, the factors have been observed and modeled individually in a laboratory experiment. However, in order to carry out all interactions between factors, a so-called full factorial design has to be observed, i.e., measurements, models, and estimations have to be done for all permutations. Due to large effort, these experiments were omitted in this thesis and are subject to future work.

Third, general queuing networks possess limitations with regards to estimation quality, especially when stateful services are present. Some mechanisms (e.g., $\beta$-nodes in the RETE graph or database tables) require the queuing network to know what has happened in the past in order to forecast the response time of the node in question. However, queuing networks are based on first-order Markov chains, i.e., they decide for the next state based on the previous state and a particular transition probability only. To enable transition probabilities into one of the subsequent states based on the entire history, hence, would result in a state explosion which cannot be simulated efficiently anymore. Therefore, the prediction quality is limited to the distributional assumption of the services.

Fourth, the application of the performance model needs preparation steps, i.e., it cannot be used out of the box. For each new case, the service times have to be derived using the presented polynomials, the resulting times have to be inserted into a tool such as $JSIMGraph$ including the selection of the correct distribution for each node, the modeling of events and probabilities, etc. Finally, the simulation has to be executed to obtain estimations for average response times and throughput. Overall, this limits the application of the performance model as it requires high technology skills to obtain a correct estimation. Hence, future work is subject to reduce the efforts of the overall prediction model application process.

8.5 APPLICABILITY

After discussing the general feasibility of GaML in Section 8.2.1 and the corresponding runtime with regards to implementation, conceptual and technical integration, reusability, and performance behavior, this section investigates the applicability of the language. As the gamification runtime environment is controlled by GaML instances, it is argued that if the language is applicable, then the runtime system is applicable as well. For this purpose, a user acceptance study was conducted to evaluate the readability and modifiability design goals of GaML to allow IT and gamification experts to work with the platform in a practical manner.

8.5.1 STUDY DESIGN

The study consisted of 36 questions clustered into seven major blocks$^{10}$. In the first block, four exogenous factors were measured: prior IT expertise ($ITEXP$) on a 5-point Likert scale, gamification expertise ($GAMEXP$) on a 5-point Likert scale, gender ($GEND$) as binary dummy variable, and age ($AGE$) in six disjoint bins (<16, 16-25, 26-35, 36-45, 46-55, >55).

The second block comprised four general gamification multiple-choice questions to record if participants are familiar with gamification concepts ($CORG$). For example, this

$^{10}$The complete questionnaire can be found in Appendix C.
block includes questions such as “How might points, levels, and badges be connected in the context of gamification?” or “What is a mission?”.

Furthermore, 17 questions were asked in blocks three to six. In these blocks, participants had to give answers to valid GaML instances with increasing complexity and difficulty. Listing 8.9 shows a simple GaML example from question block three.

```gacl
useraction actualRide {
  properties {
    carbondioxid: Decimal,
    kilometers: Decimal,
    driver: Boolean,
    rideId: Number
  }
}
```

Listing 8.9: Example listing for simple GaML question

Based on this listing, participants were asked which of the following eight statements are true:

- It defines a user action called actualRide.
- This actualRide user action has four custom properties.
- 12.2 is a correct value for kilometers.
- 12.2 is a correct value for rideId.
- 5 is a correct value for kilometers.
- 5 is a correct value for rideId.
- The field driver holds the driver’s name.
- The field driver can have the value true or false.

Participants could select multiple answers before they proceed to the next question. In this example, five questions are correct (see Appendix C). In the subsequent blocks, more complex examples and questions were given. For instance, participants were faced with the following listing:

```gacl
concept CARPOOL {
  useraction rideIntent
  useraction actualRide {
    properties { carbondioxid: Decimal, kilometers: Decimal, driver: Boolean, rideId: Number }
  }
}
point XP {
  name="Experience Point", abbreviation="XP", type=ADVANCING
  when player { did useraction rideIntent, lastsFor 24h } then { give 1 XP }
  when player { did useraction rideIntent } then { give 5 XP }
  when player { did evt1 : useraction actualRide } then { give 10+(evt1.carbonoxid/10) XP }
}
```

Listing 8.10: Example listing for complex GaML question
Based on this, the participants were asked several questions, e.g., “How many points does Bob have immediately, after 5 minutes, 60 hours, and another ride intent?” In this case, participants have to explicitly write their answers into free text inputs. Another example question based on Listing 8.9 is “Which rewards are given for which actions?”. Here, participants have to select three correct answers out of five possible ones:

- Making an actualRide with 20 (kg) saved carbon-dioxide gives the user 10 XP.
- Making an actualRide with 20 (kg) saved carbon-dioxide gives the user 12 XP.
- Creating a rideIntent gives 6 XP instantly.
- Creating a rideIntent gives 5 XP instantly and 1 more after the intent existed for at least 24h.
- Creating a rideIntent gives no points instantly and 6 after the intent existed for at least 24h.

It is important to note that participants did not receive any direct feedback on the correctness of their answers to ensure unbiased results on subjective measures at the end of the study. Moreover, participants did not get any trainings beforehand to avoid the influence of additional factors (e.g., that some participants learn the language faster than others). Overall, 76 correct answers could be given based on single choice, multiple choice, or free text input. These answers were used to measure the participants’ correctness (CORU) and to assess their understandability of the language’s abstract syntax and meta-model. Furthermore, the questionnaire also included questions were participants had to decide between different design alternatives. These questions were used to measure, if participants do not only understand the language but if they would be able to decide for correct modifications to an existing concept.

Within the seventh block, usage intentions were measured using the System Usability Scale (SUS) by [38] which is an reliable instrument (Cronbach’s α > .91) for measuring the usability of information systems [192]. The SUS questionnaire comprises ten core questions such as “I think that I would like to use this language frequently” or “I found the language unnecessarily complex” measured on 5-point Likert scales [38]. The overall SUS score for one individual is calculated from the items with

\[
SUS = 2.5 \times \sum_{i=1}^{10} q_i
\]

where \(q_i\) is the final value of \(i\)-th item from the questionnaire with \(q_i = v_i - 1\) if \(i\) is odd (with \(v_i\) being the actual rated value) and \(q_i = 5 - v_i\) if \(i\) is even in order to revert the inverse items.

In accordance with [19], the overall impression (SUSO) has been measured on an additional 7-point Likert scale (11th SUS-related question). Although SUS has been developed to assess general system usability, test questions were modified by replacing the word “system” with “language” in conjunction with [19].

The preliminary questionnaire has been pre-tested three times by five different and randomly-selected reviewers to eliminate language ambiguities and improve the overall questionnaire. In the third repetition, no additional improvements were proposed by the reviewers.
8 Evaluation

8.5.2 DESCRIPTIVE STATISTICS

The survey was completed by 42 participants (125 starts / 387 views). Six responses were removed from the sample since these participants either filled out the questionnaire 150% faster compared to the median completion time or left missing values. From the remaining 36 participants, there are 28 men and 8 women (Table 8.7).

<table>
<thead>
<tr>
<th>Gender</th>
<th>Absolute frequency</th>
<th>Relative frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>28</td>
<td>0.78</td>
</tr>
<tr>
<td>Female</td>
<td>8</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Table 8.7: Distribution of variable GEND

The average IT expertise (ITEXP) in the sample is 3.36 (1.48), prior gamification expertise (GAMEXP) is 2.86 (1.40). On average participants are between 26-34 years old (Table 8.8).

Gamification questions (CORG) were answered with 90.74% (12.88) correctness on average and 100% on median whereas GaML-related questions (CORU) were answered with an 88.2% (9.18) precision on average and 90% on median which leads to an overall correctness (CORA) of 88.51% (8.75) or 91.21% on average or median respectively. The accumulated SUS score stays at the 68.26 percentile and the overall SUS score (SUSO) averages to 4.89 (0.92). Although the questions on SUS were substituted with the word language, Cronbach’s $\alpha$ is 0.87 which accounts for a reliable measurement ($\alpha > 0.6$ is considered as reliable [59]). The null hypothesis of normality is rejected in all cases at least against a 1% significance level according to Shapiro-Wilk-Test except for the SUS measure.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>S.D.</th>
<th>Median</th>
<th>Shapiro-Wilk Test (p-value)</th>
<th>Cronbach’s $\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITEXP</td>
<td>3.36</td>
<td>1.48</td>
<td>4</td>
<td>$1.6*10^{-4}$</td>
<td>-</td>
</tr>
<tr>
<td>GAMEXP</td>
<td>2.86</td>
<td>1.40</td>
<td>3</td>
<td>$1.7*10^{-3}$</td>
<td>-</td>
</tr>
<tr>
<td>AGE</td>
<td>3.17 (26-34)</td>
<td>0.81</td>
<td>3</td>
<td>$2.7*10^{-4}$</td>
<td>-</td>
</tr>
<tr>
<td>CORG</td>
<td>90.74</td>
<td>12.88</td>
<td>100</td>
<td>$1.1*10^{-6}$</td>
<td>-</td>
</tr>
<tr>
<td>CORU</td>
<td>88.20</td>
<td>9.18</td>
<td>90</td>
<td>0.01</td>
<td>-</td>
</tr>
<tr>
<td>CORA</td>
<td>88.51</td>
<td>8.75</td>
<td>91.21</td>
<td>0.01</td>
<td>-</td>
</tr>
<tr>
<td>SUS</td>
<td>68.26</td>
<td>18.31</td>
<td>67.50</td>
<td>0.45</td>
<td>0.87</td>
</tr>
<tr>
<td>SUSO</td>
<td>4.89</td>
<td>0.92</td>
<td>5</td>
<td>$2.7*10^{-5}$</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.8: Descriptive statistics for variables

Correlations between these variables are presented in Table 8.9. In this table, correlations are marked with boldface if they are significant against a 10% level according to Pearson’s correlation test. A more detailed view of correlations can be found in the cor-
relation diagram of Figure C.1 where for each combination the correlation, significance, 
distribution, and polynomial fit is presented.

<table>
<thead>
<tr>
<th></th>
<th>SUS</th>
<th>SUSO</th>
<th>CORG</th>
<th>CORU</th>
<th>CORA</th>
<th>ITEXP</th>
<th>GAMEXP</th>
<th>AGE</th>
<th>GEND</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUS</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUSO</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CORG</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CORU</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CORA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITEXP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAM-EXP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.9: Correlation matrix of exogenous variables

Based on the significant correlations, the following statements can be concluded:

1. The higher the average SUS score ($SUS$), the higher the overall SUS ($SUSO$) rating which is consistent with [19].
2. The higher the SUS score ($SUS$), the higher the correctness of answers (CORG, CORU, CORA).
3. The higher the overall SUS ($SUSO$) score, the higher the correctness of answers (CORG, CORU, CORA).
4. The higher the correctness in one category (i.e., CORG, CORU, CORA), the higher the correctness in all other categories (CORG, CORU, CORA), i.e., having higher correctness in general gamification questions yields higher correctness in GaML understandability and vice versa.
5. The higher the IT expertise (ITEXP), the higher the correctness of answers in all categories (CORG,CORU,CORA), i.e., IT expertise helps to give more precise answers on general gamification questions as well as GaML-related questions.
6. The lower the age (AGE), the higher the correctness of answers (CORU, CORA).
7. Men (GEND) had higher IT expertise (ITEXP) in this study.
Figure 8.34: System usability scale item averages
8.5.3 INTERPRETATION AND DISCUSSION OF RESULTS

First, it is important to note that exogenous demographic variables are not correlated, except GEND and ITEXP as more men than women participated who had more IT expertise overall. In fact, the correlation diagram in Figure C.1 shows that the remaining correlations are well distributed across all boundary values. Therefore, results can be interpreted independent of AGE, ITEXP, GAMEXP, and partially GEND (i.e., without IT-EXP).

Second, results show that gamification concepts in general (90.74%) and GaML-related descriptions (88.2%) are well understood which accounts for a high understanding of GaML’s meta-model and, therefore, provides evidence that design goals DO3 and DO4 are fulfilled. Looking closer at the data reveals the following additional interpretations. For instance, older participants understand GaML worse than younger people, although this group understands gamification concepts as well as younger do. Furthermore, higher levels of IT expertise lead to higher correctness of answers in all categories. On the other side, GAMEXP has no significant influence on precision. Hence, it is concluded that still IT expertise helps to give more precise answers in general whereas expertise in gamification brings neither advantage nor disadvantage in answering the questions. This effect can probably be explained with the nature of the study as both, the questions and answers in all categories, require strong analytical and structured thinking.

Third, the SUS evaluation shows moderate usage intentions on average (68.26 percentile) with a high standard deviation (18.31 percentiles). SUS itself is reliably measured according to Cronbach’s $\alpha$ being 0.87. Figure 8.34 presents the mean and standard errors for all individual items of the SUS scores. Positive questions are ranging between 3.31 and 3.86. Participants especially think that they can learn the language quickly and that GaML is easy to use. Besides these positive results, the data also suggests improvements that are necessary for adopting the language in practical scenarios.

More precisely, the interpretation of the five inverse SUS items shows that participants primarily demand support of technical persons to use the language (2.58), that the language seems somehow complex (2.31), and that they need to learn some things beforehand (2.14). All these effects could be arguably diminished using prior trainings which were not given to participants before conducting the study. The remaining items capturing awkwardness and language inconsistencies are comparably low (< 2).

Fourth, the average SUS score (68.26) is within one standard deviation (11.6), but outside one standard error (2.6) of the mean SUS score (71.4) for a Good rating as established in [19]. This statement is further supported by the average SUSO score (4.89) which is, on the one hand, strongly correlated with the SUS score and, on the other hand, is also within one standard deviation of the Good rating in conjunction with [19]. However, adjective judgments have to be interpreted cautiously as these scales have been established for system usability studies and are not specific to any kind of formal languages.

Besides the quantitative questionnaire, participants gave qualitative feedback on the language. This feedback supports the observations further as participants with high gamification expertise demand a graphical editor as concrete syntax whereas IT experts tend to demand a more parsimonious declarative language with less textual constructs. Although the abstract syntax of GaML is well understood, it is acknowledged that for both target groups different kinds of concrete syntaxes have to be offered, e.g., a textual syntax for IT experts and a graphical syntax for gamification or domain experts (see Section 9.2).
Overall, it is concluded that GaML seems to provide an adequate means for gamification and non-gamification experts to discuss and maintain gamification concepts in a formal way and, therefore, is highly applicable for these roles in gamification projects. Since the language is automatically compilable into executable code for the developed gamification runtime environment, it is further argued that the applicability of the gamification platform itself is given implicitly. Additional ways for improving the various aspects of GaML as shown by this user study are discussed in Chapter 9.

8.6 VALIDATION OF BENEFITS

So far, the individual artifacts of this thesis have been validated individually with regards to the stated requirements (Section 4.3) or design goals (Section 5.1). Therefore, this section describes an integrative perspective across all deliverables of this thesis to validate the benefits (e.g., increase in motivational and participation, time savings for implementation) of for one real application scenario. These benefits are investigated using the SAP Customer Financial Fact Sheet application introduced in Section 8.1.

8.6.1 MOTIVATIONAL IMPACT

In [134] a mobile widget framework on top of the existing gamification runtime environment has been conceptualized and applied to reduce the efforts for the integration and visualization of gamification elements within mobile applications. Figure 8.3 (p. 120) presents the integration of some mobile widgets within the target application.

The subsequent study showed that there are significant improvements in how long subjects are engaged with the mobile application. Several engagement criteria were measured such as the number of notes and reminders participants wrote, the overall usage time of the application in minutes, and the sum of points that participants achieved. Furthermore, the SUS score has been applied again to measure subjective usability with ten items on 5-point Likert scales (Appendix C.2).

Table 8.10 presents the improvements of the gamified application (Group B) against the classical ungamified application (Group A). All of these variables are normally distributed (Shapiro-Wilk-Test) and have homogeneous variances (F-Test), except the number of written reminders. Hence, mean differences between groups A and B are tested with Wilcoxon-Whitney-Mann-Test for number of reminders and t-Test for the remaining variables.

On average, participants wrote 1.85 more reminders ($U$-value: 0, $p$-value: 0.008) and 1.9 more notes ($t$-value: 2.6, $p$-value: 0.04) to customers with gamification enabled. Furthermore, participants were engaged 8.6 minutes longer ($t$-value: 5.18, $p$-value: $8.5*10^{-4}$) with the application leading to 8.9 more experience points ($t$-value: 3.72, $p$-value: $5.9*10^{-3}$) on average. Finally, participants rated subjective usability 29.25% higher ($t$-value: 4.78, $p$-value: $2.5*10^{-3}$), i.e., usability increases by 52%. According to [19], this raises the application’s usability adjective from OK (below average) to Excellent (above average). Moreover, the presented mean differences are not only statistically significant, but Cohen’s $d$ suggests a strong effect ($> 0.8$) of this difference whereas

$$d = \frac{\mu_A - \mu_B}{\sqrt{(\sigma_A^2 + \sigma_B^2)/2}}$$

(8.14)
with \( \mu_A \) and \( \mu_B \) being the provided sample means of the groups as well as \( \sigma^2_A \) and \( \sigma^2_B \) the sample variances respectively [58]. All other statistical tests are explained in Appendix E. Furthermore, the relative improvements for each measured factor with regards to means and standard error are graphically presented in Figure 8.35.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Group</th>
<th>Mean</th>
<th>S.D.</th>
<th>W</th>
<th>F</th>
<th>( t )</th>
<th>U</th>
<th>( d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reminders (count)</td>
<td>A</td>
<td>1.75</td>
<td>0.43</td>
<td>0.009</td>
<td>-</td>
<td>-</td>
<td>0.008</td>
<td>2.64</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>3.60</td>
<td>0.89</td>
<td>0.046</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Notes (count)</td>
<td>A</td>
<td>0.50</td>
<td>0.50</td>
<td>0.12</td>
<td>0.06</td>
<td>0.04</td>
<td>-</td>
<td>1.68</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>2.40</td>
<td>1.52</td>
<td>0.49</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usage Time (minutes)</td>
<td>A</td>
<td>7.00</td>
<td>2.55</td>
<td>0.54</td>
<td>0.91</td>
<td>8.5*10^{-4}</td>
<td>-</td>
<td>3.27</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>15.60</td>
<td>2.70</td>
<td>0.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Points (count)</td>
<td>A</td>
<td>11.50</td>
<td>3.64</td>
<td>0.80</td>
<td>0.89</td>
<td>5.9*10^{-3}</td>
<td>-</td>
<td>2.35</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>20.40</td>
<td>3.91</td>
<td>0.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUS Score (percentile)</td>
<td>A</td>
<td>56.25</td>
<td>11.79</td>
<td>0.47</td>
<td>0.33</td>
<td>2.5*10^{-3}</td>
<td>-</td>
<td>3.02</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>85.50</td>
<td>6.93</td>
<td>0.65</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.10: A/B Analysis of ungamified (A) versus gamified (B) SAP Financial Fact Sheet application

Figure 8.35: Variable means with standard error of ungamified (A) versus gamified (B) SAP Financial Fact Sheet application

---

11S.D.: sample standard deviation; W: p-value of Shapiro-Wilk-Test for normality; F: p-value of F-Test for variance homogeneity; t: p-value of t-Test for mean differences; U: p-value of U-Test for mean differences; d: Cohen’s d for effect size of mean difference.
8 Evaluation

8.6.2 INTEGRATION

In order to achieve the presented results, the gamification concept was modeled in GaML. An excerpt of this application is given in Listing 8.11. For example, there are different user actions such as `playerSentReminder` or `playerCreateNote` expressing the various actions of users in the mobile application. The `customerPaid` action defines further properties signifying additional data such as the pay amount. Furthermore, point categories, levels, badges, and missions are defined. For example, `Experience Points` might be achieved in three different ways. First, they are given when the customer pays a bill within at most 14 days after a reminder has been sent or when the customer has been visited in person. While in the former case, the user’s resulting points are calculated based on the pay amount as specified by the numeric expression, in the latter case 3 points are statically given. Second, they are given upon completion of the `readHelp` mission which gives one additional `Experience Point` when reading the help for the very first time.

```plaintext
concept FFS {
    useraction playerSentReminder
    useraction playerCreatedNote
    useraction customerPaid {
        properties { daysToLastReminder: Number, visitedCustomer: Boolean, amnt: Decimal }
    }
    useraction appStarted
    useraction readHelp

    point Experience {
        name = "Experience",
        abbreviation = "XP"

        when player { did evt: useraction customerPaid, daysToLastReminder <= 14 }
            then { give 5+(evt.amnt/10000) Experience }

        when player { did useraction customerPaid, visitedCustomer = true }
            then { give 3 Experience }
    }

    point ReminderPoints {
        name = "ReminderPoints", abbreviation = "RP"

        when player { did useraction playerSentReminder }
            then { give 1 ReminderPoints }
    }

    point NotePoints {
        name = "NotePoints",
        abbreviation = "NP"

        when player { did useraction playerCreatedNote }
            then { give 1 NotePoints }
    }

    level PennyPincher { name = "Penny Pincher", threshold = 10 Experience }
    level NickelNurser { name = "Nickel Nurer", threshold = 15 Experience }
    //...
```
This GaML definition has been compiled into JSON-RPC calls compliant to the runtime’s API and deployed automatically using the runtime’s import functionality (Figure 8.36). Furthermore, the integration into the existing mobile application has been achieved through the aforementioned mobile library which can be done in three steps. First, the framework has to be globally initialized by the application where different settings such as the gamification platform’s endpoint or authentication method can be configured (Listing 8.12).}

Listing 8.12: Initialization example of mobile widgets

Second, different views might be programmatically initialized and reused to show the gamification to the end user as presented in Listing 8.13 and Listing 8.14. For example, developers may choose to reuse an entire gamification profile (Listing 8.13) including
8 Evaluation

<<component>>
Customer Financial FactSheet

<<component>>
AdminUI

<<component>>
AppViews

<<component>>
Gamification Platform

<<component>>
Gamification Widgets
Webservice

JDBCInterface

<<component>>
Database

<<component>>
MBOs

<<component>>
DataPump

<<component>>
Controllers

<<component>>
SAP ERP

SMPAPI
<<component>>
SAP Mobile Platform

JDBCInterface

<<component>>
Database

BAPI

Figure 8.36: Integration architecture for SAP Financial Fact Sheet application
several subviews for missions, badges, levels, or leaderboards, or only parts thereof
(e.g., Listing 8.14) which can be added to arbitrary views of the application. Besides
programmatic initialization, the addition of views can be also accomplished using the
drag-and-drop storyboard mechanism of the integrated development environment.
1

2

PrivateProfile * playersProfile = [[ PrivateProfile alloc] initWithFrame :self
.view. bounds ];
[self. scrollView addSubview :mView ];

Listing 8.13: Initialization of mobile player profile widget
1

2

MissionView *mView = [[ MissionView alloc] initWithFrame :frame mission :
_gData . usersProfile . missionList [i]];
[self. scrollView addSubview :mView ];

Listing 8.14: Initialization of mobile mission widget
Finally, the application has to be instrumented for the various actions that users can
do by calling the API of the runtime system (Listing 8.15).
1

[self sendRequest :@{@" method ":@" receiveEvents ", @"id":[@" submit_ "
stringByAppendingString : eventType ], @" params ":@[@[@{@" siteId ":_siteID ,@"
type":eventType ,@" playerid ":self.user ,@"data": dataDict [@"data" ]}]]}];

Listing 8.15: Example instrumentation for sending events from host application
In accordance with Figure 8.36, the mobile application itself consumes its content
from the SAP Mobile Platform, a general data management technology for mobile solutions. This platform accesses a standard ERP system for back-end data. The ERP
system’s provided interfaces were used by a so-called DataPump component which receives events from the ERP (e.g., payments received from the customer) and converts
and sends them to the gamification platform to calculate the user’s progress which is
eventually displayed in the mobile application.
168


Table 8.11: Time exposure for complete gamification enablement

<table>
<thead>
<tr>
<th>Step</th>
<th>Task</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Designing gamification concept</td>
<td>1 hour</td>
</tr>
<tr>
<td>#2</td>
<td>Formalizing and deploying gamification concept with GaML</td>
<td>0.5 hours</td>
</tr>
<tr>
<td>#3</td>
<td>Integrating gamification widgets into host application</td>
<td>4 hours</td>
</tr>
<tr>
<td>#4</td>
<td>Integrating ERP system with gamification runtime environment for back-end data</td>
<td>43 hours</td>
</tr>
</tbody>
</table>

Table 8.11 summarizes the time which has been spent in each phase. First, the designing phase took one hour since only a simple gamification concept has been proposed, i.e., only some game mechanics were utilized. To write the gamification design in GaML another 30 minutes were utilized. This time also includes the deployment within the gamification runtime system. Further four hours were spent to integrate the mobile gamification widgets into the existing application, i.e., to visualize and represent the widgets. Finally, approximately five days were necessary to implement the DataPump component and integrate it between gamification platform and ERP system.

Hence, overall approximately six days were used to accomplish a gamified version of the presented mobile application whereby the additional back-end integration utilized the most time. With regards to the artifacts developed in this thesis, however, the time spent in steps #2 and #3 is comparably low. In fact, for the other applications (e.g., the TwoGo or Networking Lunch case), in particular steps #2 and #3 took several weeks for implementation and integration of the gamification concepts, i.e., without GaML, automatic compilation, and gamification front-end widgets. Hence, it is argued that the presented artifacts accelerate these specific phases of the gamification process.

It is also important to note that the presented development times exclude the time benefits from reusing the conceptualized gamification runtime environment. Without such a platform, each gamification project would require additional effort to implement the gamification within the IS. However, since time efforts for such from-scratch implementations of gamification concepts have not been reported yet, a presentation of the benefit is subject of future work.

In the end, the integration into this sample took less than half a day for one developer unexperienced with the usage of the framework. The overall process from writing the concept with GaML until its deployment within the generic gamification platform as well as the mobile application as exemplary front-end took approximately one day. After that the application was fully operational and functional with respect to the intended gamification concept.

### 8.6.3 Conclusion

With this application scenario the validity of the thesis’ approach with respect to the gamification process (Chapter 3) could be shown. This specifically concerns the design and implementation phases as well as the artifacts that are flowing between those two.
First, a gamification concept was created which has a traceable motivational impact as hypothesized by gamification. Second, the entire gamification concept could be described and formalized using GaML in a very short amount of time. Third, the GaML instance describing the concept could be directly compiled into the underlying gamification runtime environment. Fourth, the widget framework allows for an easy and fast integration of the visual gamification aspects into the front-end, thus, enabling the a mobile application with gamification. Finally, the gamification platform allows for the immediate execution of the desired gamification concept.

Nonetheless, the results have to be interpreted cautiously and cannot be generalized for two reasons. First, the presented application and the corresponding gamification concept are not representative with regards to complexity. For example, real industry projects would require the integration with further and more complex ISs from the company's infrastructure. For example, in the presented case, the integration with external systems such as ERP still requires high integration effort in the implementation and testing phases of the gamification process. Therefore, additional engineering work is required to implement pre-integrations with external systems which would reduce implementation efforts significantly further.

Second, the gamification concept would be more sophisticated in a real-world industry project and, moreover, would require additional adaptation costs in the long run. These general costs and the potential benefits of the proposed solutions are not measured or presented herein. A subsequent longitudinal study has to be conducted in order to provide more insight into subsequent problems and the precise benefits of the artifacts developed by this thesis.

8.7 SYSTEMATIZATION

In this chapter, the artifacts of this thesis were evaluated with regards to feasibility, non-functional requirements, and benefits. In summary, the chapter illustrated the following aspects.

This chapter evaluated the feasibility of the Gamification Modeling Language by presenting its implementation artifacts, i.e., a textual editor with syntax highlighting and code completion as well as a code generator for a specific target language (Section 8.2.1). In addition, the feasibility of the gamification runtime environment was shown by presenting the concrete implementation of its components as well as the overall architecture (Section 8.2.2).

Furthermore, using these two main artifacts, five real-world applications from the industry have been gamified in a very short amount of time (Section 8.1 and Section 8.3). It is argued that this demonstrates a high degree of reusability. The statement is further supported by the fact that these gamified applications are from very different domains, hence, the runtime environment is not specific to a particular application class. Once implemented, all applications benefited additionally from the external gamification services since design changes can be flexibly incorporated at runtime.

Based on the selected SAP Network Lunch case, the performance requirements were validated since the platform is able to serve hundreds of thousands of users on a single processing node (Section 8.4). However, for larger scenarios distributed algorithms for horizontal scale-out have to be investigated. This specific consideration is, however, beyond the scope of this thesis (see Section 9.2).

The applicability of the approach was validated based on a comprehensive user study (Section 8.5). The study showed that GaML is a highly understandable language for domain and IT experts. Furthermore, domain experts are able to accomplish very simple
modifications based on the textual representation. Therefore, GaML is considered as applicable. Since the runtime environment can be controlled by the modeling language, it is argued that the runtime is applicable as well even for domain experts.

Finally, the SAP Financial Fact Sheet application was selected to validate the overall benefits (Section 8.6). Based on a user study, it could be shown that a compelling and motivating gamification scenario was implemented. Using the artifacts of this thesis, the implementation efforts to enable the gamification design could be reduced from a few weeks or several days to approximately four hours. These benefits apply only to the integration artifacts (e.g., SDKs and widget libraries) of the runtime environment, thus, excludes the direct benefits of the runtime system. Hence, the real benefits with regards to time and effort of the overall approach are arguably much larger. Since this is only a qualitative statement, future work has to analyze and quantify the total benefit.

Based on this project, it is also argued that required degree of integrability is high since developers are supported by Software Development Kits (SDKs) and widget libraries. Since these artifacts require only a few lines of code, it is argued that the desired low degree of invasivity is met as well.

Overall, these results have to be interpreted cautiously as they cannot be generalized and applied to large-scale industry projects. Nonetheless, they provide a strong indicator that developers continuously benefit from the concepts developed in this thesis.
9 SUMMARY AND OUTLOOK

This final chapter summarizes the results of this thesis and reflects them in the context of the research questions stated in Chapter 1. Based on the results and designed artifacts, further research questions and engineering activities are proposed in the outlook section.

9.1 SUMMARY

The overall research goal of this thesis was to investigate and conceptualize a generic gamification platform with high degrees of flexibility, integrability, reusability, and performance. Based on this goal, this thesis developed the following contributions to the field of gamification.

1. The definition of a precise gamification process from a software development and life-cycle perspective (Chapter 3): With the help of the provided process, researchers and practitioners understand the different tasks and roles that are involved in an ideal gamification project. Moreover, experts can analyze and understand pitfalls in the overall process. Furthermore, the process might be used to define the non-functional requirements for gamification-related technologies as those have to comply with the specific requirements determined by the surrounding process.

2. A comprehensive analysis and structure of the defining game design elements that have been translated to the domain of gamification (Chapter 4): The resulting taxonomy may help researchers and practitioners to identify appropriate elements for their gamification designs and concepts. The classification might be also helpful to structure new elements accordingly, i.e., elements which have not been translated into the gamification domain so far. Eventually, these conceptual elements can be already used at this stage to assess and compare related gamification solutions regarding their provided functionalities.

3. The formalization of the analyzed game design elements and their relationships into the gamification modeling language, a novel domain-specific language (Chapter 5): The usage of this language solved four major problems between the design and implementation phases of gamification projects. First, the language bridges the communication gap between domain and IT experts, i.e., users from both
roles can discuss gamification concepts more effectively. Second, researchers may use the language as a common means to discuss, compare, and design gamification concepts in theoretical work. Third, it helps to rigorously assess gamification technologies with regards to their functional features, i.e., to proof if a technology is compliant with GaML or a certain subset thereof. Finally, the language helps IT experts to speed up the implementation process since GaML is automatically compilable into runtime environments for corresponding gamification technologies.

4. A derivation of solution classes for related gamification technologies (Chapter 6): This thesis introduced a distinction of existing gamification solutions into achievement systems, integration gamification solutions, and generic gamification platforms based on their non-functional properties. These classes and the description of their inherent structures and behaviors may help researchers and practitioners who are in charge of planning, executing, and operating the gamification project to select the technology or approach that fits best to the project’s requirements.

5. The design, implementation, and evaluation of a novel and generic runtime environment for gamification concepts (Chapter 7): Since the analysis of related work based on GaML’s elements yielded that none of the existing runtime environments suffices all functional and non-functional requirements, the necessity for a new platform approach has been demonstrated. The developed runtime environment fulfills all requirements determined by GaML as well as all derived non-functional requirements.

6. An automated compilation procedure of GaML into executable code for the proposed runtime environment (Chapter 7): Besides the practically relevant speed-up of development and maintenance activities, the approach also demonstrates exemplarily how GaML or a subset thereof can be transformed into target code or models for arbitrary gamification runtime environments.

Besides these general contributions, four concrete research questions have been formulated. In the following, each research question is reflected by briefly summarizing the concrete results and findings within this thesis.

**RESEARCH QUESTION 1: WHICH GAMIFICATION CONCEPTS, STRUCTURES, AND RELATIONSHIPS EXIST AND HAVE TO BE PROVIDED BY A GENERIC GAMIFICATION PLATFORM?**

Based on a comprehensive review of the existing gamification research literature (Chapter 4), game mechanics were surveyed which have been isolated from games and applied in non-game contexts by gamification researchers and practitioners. These elements have been structured in accordance with the general game design hierarchy by [65] and provided the functional foundations for the rest of the thesis.

Overall, 19 elements were identified on the first game interface pattern (L1) level (Table 4.1 and Table 4.2). Further eight coarse-grained conceptual requirements were identified with regards to the second game design pattern and mechanics (L2) level of Deterding’s taxonomy (Section 4.2.2) which have been decomposed into 26 core elements in Chapter 5. Each gamification concept has been described and disambiguated by defining its concrete name, providing a comprehensive definition from literature, outlining possible synonyms and subtypes, and referring to the corresponding citations.
In addition, relationships (e.g., associations, aggregations, compositions) between the presented elements were described.

As the review covered the most important gamification literature and another more recent review in [91] did not yield any novel elements beyond the considered ones, this research question is answered appropriately.

However, the requirements are subject to change and might be potentially incomplete (cf. Section 9.2). Nonetheless, future modifications and enhancements can be easily incorporated into the proposed schema of the framework.

**RESEARCH QUESTION 2: HOW CAN THESE CONCEPTS BE REPRESENTED IN A FORMAL WAY WHICH IS, NONETHELESS, UNDERSTANDABLE FOR GAMIFICATION AND DOMAIN EXPERTS?**

Because of a missing taxonomy there was also no formal mechanism in place to describe gamification concepts in a rigorous manner. Consequently, appropriate tool support for expressing gamification concepts and proofing its correctness was missing, e.g., to validate whether a particular instance complies with the specified meta-model and static semantics.

Therefore, this thesis proposed GaML as a formal domain-specific language (Chapter 5). The current proposal of GaML’s abstract syntax reflects all concepts and relationships identified under research question one. Furthermore, a textual representation was chosen as the concrete syntax.

Besides its formality, the language was designed as domain-specific language to be at least understandable for IT and domain experts. Moreover, domain experts should be able to do simple modifications based on an existing instance. The subsequent user study showed that participants who have never seen GaML before are able to correctly answer understandability and modifiability questions with a 90% precision on average. As the study collected data across all levels of gamification and IT expertise, this positive statement applies to both groups equally. Based on the results of the study, it is argued further that this also validates the logical coherence of the presented requirements and taxonomy of research question one.

Moreover, the evaluation chapter presents appropriate tool support for writing instances of GaML, i.e., domain or gamification experts are supported through visual highlighting of the language’s keywords, code completion for terminals, and immediate presentation of syntactic and semantic errors (Chapter 8).

Overall, the design goals of GaML in particular and the research question in general are considered to be fulfilled.

**RESEARCH QUESTION 3: WHICH SERVICES, COMPONENTS, AND STRUCTURES ARE NECESSARY TO CONSTITUTE A GENERIC PLATFORM FOR GAMIFICATION WHICH SUPPORTS ALL IDENTIFIED CONCEPTS OF RESEARCH QUESTIONS ONE AND TWO?**

Based on elements dictated by GaML, it has been found that none of the 29 analyzed solutions suffices all conceptual and functional requirements (Chapter 6). The related technologies have been classified into three main solution classes determining the supported requirements. Based on the non-functional requirements, it has been argued that one class, namely the Generic Gamification Platform (GGP) class, can be considered as appropriate for enterprise gamification projects. However, all three analyzed solutions within this class do not fully comply with the formalized requirements (Chapter 6).
Therefore, a novel runtime environment has been proposed in Chapter 7 based on service-oriented and event-driven principles from the corresponding architecture paradigms. In itself, the runtime system is a hybrid architecture type comprising a business entity provider fulfilling persistence and transactional requirements and an event processor fulfilling at least soft real-time event processing. Both components interact and communicate with each other by the means of events in different communication modes.

Using the architecture’s APIs, clients may interact with this runtime environment. By describing the translation and mapping of GaML elements into corresponding calls of the runtime system, it has been shown that the platform and its APIs support all desired functional requirements. As useful side-effect for software engineers and developers, valid instances of GaML can be directly compiled into code for the presented runtime, i.e., the gamification model can be directly executed within the runtime environment.

Overall, this research question is considered to be answered since the proposed gamification runtime environment supports all surveyed and formalized requirements as presented in the evaluation.

**RESEARCH QUESTION 4: HOW CAN THE IDENTIFIED SERVICES AND COMPONENTS BE SEAMLESSLY INTEGRATED WITH ARBITRARY ENTERPRISE INFORMATION SYSTEMS?**

The overall platform has been designed to be consumable as a service, i.e., clients may use standard means to access the API, e.g., issuing RPCs using REST, SOAP, or JSON-RPC interfaces. Based on these services, five real-world applications used in enterprises could be gamified in a short amount of time (Chapter 8). It is argued that this demonstrates a high degree of flexibility and reusability of the approach in general.

Using only the low-level services (e.g., REST or SOAP), however, requires a lot of time for the integration of the gamification runtime with the host application, e.g., to instrument the information system sending events or to present the tangible gamification elements on the front-end. Therefore, software development kits and reusable graphical widgets for various target platforms have been introduced in Chapter 8. These widgets provide a convenient mechanism to work with the platform’s services on a more coarse-grained level of integration. With these additional components, the integration effort could be significantly reduced, e.g., from multiple days (when implemented from scratch) to four hours for the integration into one of the selected real enterprise mobile applications (Section 8.6).

Besides aspects such as high flexibility and reusability, performance behavior, i.e., feedback in soft real-time, was considered as very important. Therefore, based on the platform prototype a comprehensive performance analysis has been conducted. This analysis comprised two steps.

First, a case-based performance comparison between different architecture alternatives has been carried out. For example, it has been shown that the asynchronous architecture alternative is faster by seven orders of magnitude compared to the pure synchronous case. It could be shown that the proposed extensions for managing persistent state data within event-driven technologies (e.g., CEP) introduce just a marginal performance overhead. In contrast to existing event-driven systems this allows the effective encapsulation of state data and ex post analytics.

Second, to generalize from the case-based performance, a generic performance model has been introduced using queuing networks and a relevant selection of factors which have an influence on the system’s overall performance. The presented model
allows for a prediction accuracy between 85% and 90% according to MRE. Hence, the model is suited for practitioners to forecast the system’s performance based on estimated or measured parameters such as the event workload or the number of gamification rules. Vice versa, the model can be used during planning to precisely calculate the capacity and to correctly provision the underlying infrastructure.

Based on these performance models it has been shown that the runtime environment may support hundreds of thousands parallel users even on a single instance depending on parameters determined by the gamification concept or the underlying infrastructure.

Since the proposed artifacts could significantly reduce the implementation and integration efforts for gamification with a low degree of invasivity and high degree of performance as required by the surrounding gamification process, this research question is considered to be appropriately answered.

9.2 OUTLOOK

Although this thesis solved major problems with regards to an effective and efficient gamification process (Chapter 3) or the state-of-the-art technologies (Chapter 6), subsequent questions and problems emerged out of the solved ones. Those are summarized in the following starting with questions related to the introduced GaML and ending with additional questions for the gamification runtime environment.

9.2.1 TAXONOMY COMPLETENESS

The entire stack from meta-model, methods, and technical solutions proposed in this thesis is, implicitly, merely as complete as the initial and surveyed requirements of Chapter 4 are. Hence, a change in the requirements has to be potentially reflected in all proposed artifacts, i.e., the GaML specification, the compiler, the runtime environment, and the integration artifacts. Therefore, to drive the development of GaML further, a lively discussion in academia and industry has to be initiated that eventually may lead to a standardization of the language.

For this purpose, game mechanics have to be surveyed beyond the existing research literature, e.g., by reviewing additional industry examples or non-research publications, as provided herein. For example, other kinds of game mechanics or relationships between can be observed as requirements in practical projects and which are not provided by GaML yet. This list of additional elements includes, for example, virtual items that are connected to a level progress structure, arbitrary conditions for avatar levels rather than just connecting point thresholds to them, or further elements such as virtual rooms or redeemable items (e.g., [43]).

9.2.2 OPERATIONAL SEMANTICS OF GAML

The dynamic semantics of GaML have been described as translational semantics, i.e., based on a particular target language and, furthermore, a certain computational model (i.e., rule-based processing). Hence, the semantics are closely coupled with the utilized runtime technologies.

However, to abstract further from the target code and specifics of the implementation, the dynamic semantics should be defined with a well-defined mechanism, for example, using operational or denotational semantics [82]. As this would enable the abstraction from a specific computational model, it is argued that this could help to
establish the language further. For example, the specification could be used by other vendors to propose other kinds of gamification runtimes (e.g., runtimes based on an imperative computational model) that comply with the operational semantics and, thus, fully support the intended concepts of the source language.

9.2.3 GRAPHICAL RULE EDITOR

In this thesis, GaML’s concrete syntax has been proposed to be textual. However, the corresponding study shows that this textual representation negatively correlates with the participants’ usage intentions of the language, despite the fact that they understand it well. For example, in the study people reported that they may need additional trainings, otherwise they may refuse to use it.

Nonetheless, based on the qualitative feedback that was gathered, it can be concluded that representing the language with a graphical concrete syntax such as a form-based graphical editor may increase adoption and usage intentions.

Hence, it is proposed that a graphical editor based on the full presented meta-model or a subset thereof is created and compared with the textual form in a controlled experiment. The working hypothesis here would be that the graphical editor increases the various aspects with regards to ease of use and usage intentions without harming understandability and modifiability.

Furthermore, this thesis has not studied the benefits of GaML compared to the target language, i.e., DRL herein. Another experiment could, therefore, show how much GaML improves the understandability and modifiability compared to a technical language which has to be used without GaML being in place.

Finally, the consideration of GaML’s writeability has been completely omitted in this thesis, i.e., IT experts and, more specifically, domain experts are able to write valid GaML instances on their own. In general domain-specific languages are not intended to be writeable for domain experts [82]. However, having a graphical editor as concrete syntax for GaML in place would allow studying this specific property as well. In particular, the working hypothesis would be that domain experts are at least able to write simple gamification concepts on their own using a graphical editor.

9.2.4 AUTOMATIC DERIVATION OF BUSINESS ENTITY PROVIDER SEMANTICS

Given the proposed architecture of the runtime environment in asynchronous mode, a so-called BEP proxy component was introduced to manage the state information locally within the EPA (Section 7.3.3). It has been argued that this proxy has to provide exactly the same interfaces including syntax and semantics as the master BEP plus handling and synchronizing the messages in between.

In practice, the major disadvantage of this approach is that it is cumbersome to implement on the one hand and hard to maintain over the system’s life-time on the other hand.

More precisely, on the first implementation, the semantics of the interfaces’ methods have to be implemented twice, i.e., once for the real BEP and once for the BEP proxy. Although syntax and semantics are required to be equal, both components may use different implementation approaches to handle the data. For example, the real BEP may use a database connection to persist the data, while the BEP proxy uses internal data structures for efficiency purposes and has to deal with the message replication\textsuperscript{1}.

\textsuperscript{1}Of course, it is conceivable that the BEP proxy may use an in-memory database and, thus, may reuse the implementation. However, this applies only to the data management and does not include the message passing functionality.
9.2 Outlook

More general, the BEP and the BEP proxy could be even based on different technologies or written in different programming languages.

Furthermore, the application developer has to ensure that every change in the master BEP has to be reflected in the BEP proxy as well. While inconsistencies in the syntax of interface implementations might be already detected at compile-time, changing the semantics may easily introduce inconsistent behaviors between the different implementations.

Hence, it is proposed to investigate an automatic procedure that derives the BEP proxy implementation from a general BEP’s specification. This could be done, for example, in a model-driven way where a common meta-model is used to generate both implementations of the BEP, potentially based on different programming languages and technology platforms, automatically.

9.2.5 IMPROVED SYNCHRONIZATION STRATEGIES FOR TRANSACTIONAL PROCESSING

In addition to the inconsistency problems regarding syntax and semantics, it has been highlighted in Section 7.3.3 that the approach is not scalable without additional effort. This concerns, for example, having one event processor instance per user, i.e., multiple EPAs, which are accessing and modifying the data concurrently. For this purpose transactional mechanisms have to be investigated and proposed for rule-based and event-based systems. However, simply translating optimistic locking strategies as known from database technologies would require the introduction of rollback mechanisms and, thus, requires to hold events and other data longer within memory than necessary for the business logic. Furthermore, while a rollback takes places, event processing has to be stopped or at least deferred. However, these mechanisms contradict the performance requirements of real-time event processing.

These problems and additional questions regarding horizontal scalability have to be answered to allow for a flexible and independent distribution of the proposed hybrid architecture.

9.2.6 ADDITIONAL ENGINEERING ASPECTS

The gamification platform has been considered as a generic component in this thesis. Hence, some implementation details were omitted which have been either partially implemented in the thesis’ prototype only or are at least in preparation. The following list compiles a non-exhaustive list of additional features which are necessary for a real and productive gamification service.

First, the BEP and the EPA may support some kind of plug-in mechanisms that allows others (e.g., third parties) to extend the current functionality, e.g., by adding new entities and domain concepts, new or modified API interface definitions, or new core rules and events. Furthermore, the plug-in mechanism may allow to exchange specific technologies, e.g., a certain rule engine or persistency manager.

Second, multi-tenancy considerations have not been focused in this thesis. Taking the current approach, each tenant (i.e., customer with exactly one gamification concept) would require its own distinguished instance of the platform. However, with regards to resource utilization it may be more efficient that multiple tenants share the same physical system. For this purpose, additional technology-specific concepts are necessary to investigate. While the BEP might be simply enabled for multiple tenants by introducing an additional tenant column in the database schema, there are equivalent concepts...
for handling multi-tenancy in rule-based or event-driven technologies yet that consider the nature of gamification rules and events. Additionally, the tenant notion might be extended to gamification concepts, i.e., one customer may want to consolidate multiple disjoint gamification concepts under the same instance. This would require the implementation and management of a two-dimensional tenant structure.

Third, security mechanisms within the gamification runtime environment have only been partially considered and, therefore, require additional improvements. For example, in Chapter 7 a global visibility flag per avatar has been proposed, i.e., to let the user decide whether he or she wants to be visible to others or not. However, in practical enterprise scenarios it would be necessary to configure the visibility and accessibility of all gamification elements individually, i.e., the player may decide that he shows some of the badges but not the points. For this configuration it is also necessary to reflect that there are interdependencies between the elements.

Finally, the widgets proposed for fast front-end integration have been presented in Section 8.2.2.5 as fixed software components, i.e., they are fixed with regards to the presented content (e.g., collection of badges), the supported technologies (e.g., JavaScript), and their look and feel (e.g., only customizable via style sheets). For example, there is exactly one widget for notifications, user-accepted mission, etc.

However, for practical scenarios it would be useful to model custom widgets in the back-end first. For example, domain experts may design individual graphical components with their own look and feel comprising a set of specific gamification elements. The resulting design model is stored independent of the final technology platform. Furthermore, these models can be used to automatically generate technology-specific widgets which can be integrated and consumed by the front-end instead of the pre-determined widgets. This would allow for more flexible gamification widgets beyond the ones proposed by this thesis.
BIBLIOGRAPHY


Bibliography


A GAML ELEMENTS

This appendix presents the entire specification of GaML’s grammar. This also includes selected examples as presented in Chapter 5.

A.1 TERMINALS

DATATYPE

Boolean

Text

DATATYPE

NUMTYPE

Number

Decimal

NUMTYPE

DATATYPE

Date

Time

Figure A.1: Terminal rules
Figure A.2: Terminal rules 2
Figure A.3: General numeric expressions
Figure A.4: Operand types
A.2 L1 CONCEPTS

Model

Element

GameLevel

Point

Skill

Mission

Role

Leaderboard

Level

Good

Badge

Event

Figure A.5: Space syntax (as shown in Figure 5.1)
A. GaML Elements

Figure A.6: Game level syntax (as shown in Figure 5.2)
A.2 L1 Concepts

Event

EventClass

EventBody

InverseEvent

EventData

Field

Badge

Figure A.7: Event syntax

Figure A.8: Achievement syntax
Point

```
point ID { name = STRING, 
  default, 
  description = STRING, 
  abbreviation = STRING, 
  internal, 
  type = POINTTYPE } 
```

```plaintext
POINTTYPE

ADVANCING
  REDEEMABLE
  KARMA
  SKILLPOINT
  AUXILIARY
  REPUTATION [ INT, INT ]
```

Figure A.9: Point syntax
Mission

```
mission ID {
    name = STRING,
    description = STRING
    available When
    initiatedBy = rule user
}
```

Figure A.10: Mission syntax
**Skill**

```
skill ID {} 

name = STRING 

, description = STRING 

, image = STRING 

, benefit = STRING 

, point = ID 

, available for TIME 
```

Figure A.11: Skill syntax

**Level**

```
level ID {} name = STRING 

threshold = NUMERICOPERAND ID 

, image = STRING 
```

Figure A.12: Level syntax
Good

```
ID
{ good
  name = STRING, description = STRING,
  image = STRING, GoodType
  price = NUMERICOPERAND, ID
}
```

Figure A.13: Good syntax

Role

```
ID
{ role
  name = STRING,
  skills
    { ID
    }, ID
  description = STRING,
  image = STRING
}
```

Figure A.14: Role syntax
Leaderboard

```plaintext
leaderboard [ID] {
  name = STRING,
  point = ID,
  aggregation = AGGREGATION,
  order = ascending, descending,
  assembles = player, team
}
```

Figure A.15: Leaderboard syntax
A.3 CONDITION ELEMENTS

Rule

When

Then

When

when

PlayerEntity

{ }

validFrom = DATE

validTo = DATE

PlayerOr

TeamEntity

{ }

validFrom = DATE

validTo = DATE

TeamOr

PlayerEntity

player

EntityVariable

TeamEntity

team

EntityVariable

EntityVariable

ID

Figure A.16: General condition syntax (as shown in Figure 5.3)
Figure A.17: Boolean constraint and conditions syntax (as shown in Figure 5.4)

Figure A.18: Player condition syntax (as shown in Figure 5.4)
A.3 Condition Elements

TeamOr

TeamAnd

TeamAnd

TeamAnd

TeamTopElem

TeamTopElem

TeamTopElem

TeamAtom

TeamAtom

TeamNegation

TeamNegation

not TeamTopElem

Figure A.19: Boolean constraint and conditions syntax

TeamCondition

Condition

has PlayerEntity who {PlayerOr}

Figure A.20: Team condition syntax (as shown in Figure 5.5)
EventDef

EventCondition

EventClass

EventCondition

Figure A.21: Event condition syntax
A.3 Condition Elements

**Figure A.22:** Event condition right term syntax

**Figure A.23:** Condition references syntax
A. GaML Elements

**BadgeRef**

```
 BadgeRef
   badge
   ID
```

**LevelRef**

```
 LevelRef
   level
   ID
```

**ItemRef**

```
 ItemRef
   good
   ID
```

**MissionRef**

```
 MissionRef
   mission
   MissionItem
```

**MissionItem**

```
 MissionItem
   ID
   available
   completed
```

Figure A.24: Condition item references syntax

**SkillRef**

```
 SkillRef
   skill
   SkillItem
```

**SkillItem**

```
 SkillItem
   ID
   level
   INT
   active
```

Figure A.25: Skill condition syntax

**PointRef**

```
 PointRef
   point
   ID
   AGGREGATION
   COMPARATOR
   NUMEXPR
```

Figure A.26: Point condition syntax

**Luck**

```
 Luck
   luck
   P
   NUMEXPR
```

Figure A.27: Random condition syntax
A.3 Condition Elements

LocationDef

```
location (lat COMPARATOR NUMEXPR ,
           long COMPARATOR NUMEXPR )
```

Figure A.28: Location condition syntax

Leader

```
top (to = NUMEXPR ,
     from = NUMEXPR ,
     leaderboard = ID )
```

Figure A.29: Leader condition syntax

RoleRef

```
role ID
```

Figure A.30: Role condition syntax
Figure A.31: Event condition syntax
A.4 CONSEQUENCE ELEMENTS

Then

Figure A.32: General consequence syntax (as shown in Figure 5.6)
EventRefCons

create event ID
delete

\( \text{ID} = \text{NUMEXPR} \)

\( \text{EQUALITYTYPES} \)

EntityCond

ID

PointRefCons

give \( \text{NUMEXPR} \)
remove \( \text{NUMEXPR} \)
set \( \text{NUMEXPR} \)

\( \text{ID} \) to EntityCond
\( \text{ID} \) from EntityCond
\( \text{ID} \) for EntityCond

Figure A.33: Event consequence syntax

Figure A.34: Point consequence syntax
A.4 Consequence Elements

**Notification**

\[
\text{notify} \rightarrow \text{STRING} \rightarrow \text{to} \rightarrow \text{EntityCond} \\
\text{message} \rightarrow \text{STRING} \rightarrow \text{title} \rightarrow \text{STRING} \\
\text{image} \rightarrow \text{STRING} \\
\text{type} \rightarrow \text{STRING} \\
\text{to} \rightarrow \text{EntityCond} \\
\text{Figure A.35: Notification consequence syntax}
\]

**Narration**

\[
\text{narration} \rightarrow \{ \text{name} \rightarrow \text{STRING}, \text{description} \rightarrow \text{STRING} \} \\
\text{recipient} \rightarrow \text{EntityCond} \\
\text{image} \rightarrow \text{STRING} \\
\text{video} \rightarrow \text{STRING} \\
\text{audio} \rightarrow \text{STRING} \\
\text{Figure A.36: Narration consequence syntax}
\]
Figure A.37: General consequence syntax
B EXCERPT FROM API SPECIFICATION

In this appendix, excerpts from the interfaces which can be utilized by the client are given.

B.1 IADMINAPI

```java
public interface IAdminAPI {
    long create(String gpType, Map<String, Object> keyValuePairs) throws Exception;
    long update(String gpType, long ID, Map<String, Object> keyValuePairs)
        throws Exception;
    long createApp(String name, String description, String owner);
    boolean deleteApp(String name);
    boolean deleteAllPlayers();
    boolean deleteAllTeams();
    long createPoint(String name, String abbreviation, PointType pointType);
    long createPoint(String name, String abbreviation, boolean isInternal, PointType pointType);
    boolean deletePoint(long id);
}
```

B.2 IQUERYAPI

```java
public interface IQueryAPI {
    Player getPlayer(String playerID);
    Avatar getAvatar(String avatarID);
    Team getTeam(String teamID);
    Mission getMission(String id);
    Badge getBadge(String badgeName);
```
```java
B Excerpt from API Specification

boolean hasAvatarBadge(String avatarID, String badgeName);
Collection<Badge> getBadgesForAvatar(String avatarID);

boolean hasAvatarLevel(String avatarID, String badgeName);
Level getLevelForAvatar(String avatarID, String pointName);
Collection<Level> getAllLevelsForAvatar(String avatarID);

Collection<Mission2Avatar> getMissionsForAvatar(String avatarID);
boolean hasAvatarMission(String avatarID, String missionName, MSTATUS status);

Score getPointsForAvatar(String avatarID, String pointName, AGGREGATION agg);

boolean hasTeamBadge(String teamID, String badgeName);
boolean hasTeamLevel(String teamID, String badgeName);
boolean hasTeamMission(String teamID, String missionName, MSTATUS status);
Collection<Mission2Avatar> getMissionsForTeam(String teamID);

B.3 IUPDATEAPI

public interface IUpdateAPI {
    Player createPlayer(String playerID);
    Avatar createAvatar(String avatarID, String playerID);

    boolean givePointsToAvatar(String avatarId, String pointType, double amount);
    boolean givePointsToAvatar(String avatarID, String pointType, double amount, String reason);

    boolean addMissionToAvatar(String avatarID, String missionName);
    boolean completeAvatarMission(String avatarID, String missionName);
    boolean rejectMissionForAvatar(String avatarID, String missionName);

    boolean addBadgeToAvatar(String avatarID, String badgeName);
    boolean addBadgeToAvatar(String avatarID, String badgeName, String reason);

    boolean activateSkill(long avatarId, String skillName);

    boolean addAvatarToTeam(String avatarID, String teamID);
    boolean deleteAvatarFromTeam(String avatarID, String teamID);

    boolean givePointsToTeam(String avatarId, String pointType, double amount);
    boolean givePointsToTeam(String teamID, String pointType, double amount, String reason);

    boolean addMissionToTeam(String teamID, String missionName);
    boolean completeTeamMission(String teamID, String missionName);
    boolean rejectMissionForTeam(String teamID, String missionName);

    boolean addBadgeToTeam(String teamID, String badgeName);
}
B.4 IAnalyticsAPI

```java
public interface IAnalyticsAPI {
    List<XYValue> getAnalyticsQuery(Date start, Date end, AGGREGATION agg, String metric, String groupByString, String filter);
    List<XYValue> getAnalyticsQuery(Date start, Date end, AGGREGATION agg, String metric, String groupByString, String filter1, String filter2);
    List<XYValue> getLagQuery(Date start, Date end, AGGREGATION agg, String metric, String groupBy, String site, int lagAmount);
}
```

B.5 IUSERCONFIG

```java
public interface IUserConfig {
    boolean changePublicFlag(String avatarID, boolean isPublic);
    boolean updateAvatarName(long avatarId, String avatarname);
    Collection<UserConfig> getUserConfig(long avatarId);
    boolean updateUserConfig(long avatarId, String key, String value);
    boolean setShowcaseBadge(String avatarID, String badgeName, int position);
}
```

B.6 IRULESERVICE

```java
public interface IRuleService {
    Collection<Rule> getAllRules();
    Rule getRule(long id);
    boolean deleteRule(long id) throws RuleRuntimeException;
    long createRule(String name, String description, String when, String then, String delayClause, Integer priority, Date validFrom, Date validTo) throws RuleRuntimeException;
    boolean deployAllRules() throws RuleSyntaxException;
    boolean deployRule(long ruleId) throws RuleSyntaxException;
}
```
B.7 IEVENTMANAGER

```java
public interface IRuleService {
    boolean receiveEvents(EventObject[] obj);
    boolean resetRuleEngine();
}
```
C GAML STUDY

This appendix contains the artifacts and supplementary materials to the GaML study of Section 8.5.

C.1 QUESTIONNAIRE

In this section the complete questionnaire which was used for the user study of GaML is presented. As described in Section 8.5, the user study consists of seven blocks which are outlined below. For each block, the GaML example and corresponding questions are shown. The correct answers are marked with italics.

C.1.1 BLOCK 1

IN01: ASSESS YOURSELF IN TERMS OF IT KNOWLEDGE AS WELL AS GAMIFICATION KNOWLEDGE [SINGLE-CHOICE]

- IT-Experience (e.g., Programming Languages, SQL, Development) (1-5 Scale)
- Gamification Experience (1-5 Scale)

IN02: HOW OLD ARE YOU? [SINGLE-CHOICE]

- < 16
- 16-25
- 26-35
- 36-45
- 46-55
- > 55

IN03: WHAT IS YOUR GENDER? [SINGLE CHOICE]

- Female
- Male
C.1.2 BLOCK 2

GC01: WHAT WOULD YOU EXPECT, WHEN READING THE TERM MISSION? A MISSION ... (MULTIPLE-CHOICE)

- ... can be completed by performing specified tasks.
- ... is completed when a single task is done.
- ... can give the user certain things (badges, points, etc.) upon completion.
- ... can be linked to other missions, so that these missions are only available after the mission is completed.
- ... can be composed of several other missions and all of them have to be completed, to complete the main mission.
- ... only has a single reward for the user, e.g. giving him a badge or some points.

GC02: WHAT WOULD YOU THINK A USER ACTION DESCRIBES? (MULTIPLE-CHOICE)

- ... does something to or with the user.
- ... is triggered when a user does or did something.
- ... creates a new user.

GC03: WHAT MIGHT AN INTERNAL EVENT BE? (MULTIPLE-CHOICE)

- ... is only seen by a single user, handling his or her internal information.
- ...is not seen by any user and is only used internally for handling certain information.
- ...is used by programmers temporarily, to write down an event before they think of a better name.

GC04: HOW MIGHT POINTS, LEVELS, AND BADGES BE CONNECTED IN THE CONTEXT OF GAMIFICATION? (MULTIPLE-CHOICE)

- Badges can be bought using points. Levels depend on the number of badges you have.
- Levels are reached by having at least a given amount of points. Badges are independent of both.
- A user reaches a certain level, as soon as he has enough points. Badges may additionally be awarded for levels, points or other achievements.
- Badges are gained for specific point amounts. Levels are independent from another.
- All three are completely independent from another.
C.1.3 BLOCK 3

GS02: WHAT DO YOU EXPECT THIS PART OF THE CODE TO DO? (MULTIPLE-CHOICE)

```
useraction actualRide {
  properties {
    carbondioxid:Decimal,
    kilometers:Decimal,
    driver:Boolean,
    rideId:Number
  }
}
```

Listing C.1: GaML example code for question block 3.1

- It defines a user action called actualRide.
- This actualRide user action has four custom properties.
  - 12.2 is a correct value for kilometers.
  - 12.2 is a correct value for rideId.
  - 5 is a correct value for kilometers.
  - 5 is a correct value for rideId.
- The field driver holds the driver’s name.
- The field driver can have the value true or false.

GS02: WHAT DO YOU EXPECT THIS PART OF THE CODE TO DO? (MULTIPLE-CHOICE)

```
point XP {
  name="Experience Point",
  abbreviation="XP",
  type=ADVANCING
}
```

Listing C.2: GaML example code for question block 3.2

- It creates a new point category called XP.
- The point category is called Experience Point.
- It is a point category, which is used to represent a player’s progress.
- The created point category has an abbreviation “XP” and a long name “Experience Point”.
- It gives a user a point in the category XP.
GS03: WHAT DO YOU EXPECT THIS PART OF THE CODE TO DO? (MULTIPLE-CHOICE)

```java
when player {
  has point XP, SUM > 3
}
then {
  give badge IceBreaker
}
```

Listing C.3: GaML example code for question block 3.3

- The when block has to be true, so the then block is executed.
- The then block has to be true, so the when block is executed.
- The Ice Breaker badge is given, if the user has 3 or more points in a random point category.
- When the Ice Breaker badge is earned, the user’s XP points are set to at least 3.
- As soon as the user has at least 3 XP points, he or she receives the Ice Breaker badge.

GS04: WHAT DOES THE WHOLE CODE BLOCK DEFINE? (SINGLE-CHOICE)

```java
concept CARPOOL {
  useraction rideIntent
  useraction actualRide {
    properties { carbondioxid:Decimal, kilometers:Decimal, driver:Boolean, rideId:Number }
  }
  internalevent socializerEvent {
    properties { userid:Text, friend:Text }
  }
  point Socializer { name="Socializer Point", abbreviation="SP" }
  point XP { name="Experience Point", abbreviation="XP", type=ADVANCING }
}
```

Listing C.4: GaML example code for question block 3.4

- It defines a new Gamification concept called “concept”.
- It defines a new Gamification concept called “CARPOOL”.
- It defines a new Gamification concept without any specific name.
- It defines a new Gamification concept called “socializerEvent”.

GS05: HOW ARE THE EVENTS OF THIS CODE DEFINED? (MULTIPLE-CHOICE)

- There are two user events and one internal event.
- “Santa Claus” is a possible value for the property “friend” in socializerEvent.
- The useraction rideIntent does not have any custom properties.
- The only non-numerical property of actualRide is “driver”.
- There are three user events.
GS06: WHAT DO YOU THINK IS EXPLICITLY KNOWN ABOUT EACH "ACTUAL RIDE" THAT USERS ARE DOING? TRY NOT TO "READ BETWEEN THE LINES" OR FIND IMPLICIT INFORMATION (E.G. THOUGH THERE IS AN EVENT WITH DRIVER ID AND FRIEND ID, ONE DOES NOT EXPLICITLY KNOW, WHETHER THIS FRIEND WAS ON THE SAME RIDE OR NOT.) [MULTIPLE-CHOICE]

- ... how much carbon dioxide the user saved with the ride.
- ... how old the user is.
- ... whether the user was a passenger.
- ... whether the user was the driver.
- ... how many people shared the ride.
- ... which is the user’s id.

C.1.4 BLOCK 4

```
concept CARPOOL {
  useraction rideIntent
  
  useraction actualRide {
    properties { carbondioxid:Decimal, kilometers:Decimal, driver:Boolean, 
    rideId:Number }
  } 

  point XP {
    name="Experience Point", 
    abbreviation="XP", 
    type=ADVANCING 

    when player { did useraction rideIntent, lastsFor 24h } 
    then { give 1 XP } 

    when player { did useraction rideIntent } 
    then { give 5 XP } 

    when player { did evt1 : useraction actualRide } 
    then { give 10+(evt1.carbondioxid/10) XP } 
  }
}
```

Listing C.5: GaML example code for question block 4

GA01: WHAT IS TRUE ABOUT THE POINT DEFINITION? [MULTIPLE-CHOICE]

- It defines a point category called XP with long name "Experience Point".
- It defines a point category called "Experience Point".
- XP points can be gained in four different ways.

GA02: WHICH REWARDS ARE GIVEN FOR WHAT? [MULTIPLE-CHOICE]

- Making an "actualRide" with 20 (kg) "carbon oxid" saved, the user receives 10 XP points.
• Making an "actualRide" with 20 (kg) "carbonoxid" saved, the user receives 12 XP points.

• Creating a rideIntent gives 6 XP points instantly.

• Creating a rideIntent gives 5 XP points instantly and 1 more after the intent existed for at least 24h.

• Creating a rideIntent gives no points instantly and 6 after the intent existed for at least 24h.

GA03: BOB IS A USER OF THE GAMIFIED CARPOOL APPLICATION. HE HAS JUST CREATED HIS ACCOUNT AND HIS XP ARE CURRENTLY 0. [FREE-TEXT]

• After creating his first rideIntent he has 5 XP points.

• After the rideIntent was in the system for 60h he has 6 XP points.

• His ride has taken place and the actual Ride event was triggered. Without knowing how much carbon he saved, he has to have at least 16 XP points by now.

• The actualRide saved 50kg of carbon. Consequently, Bob has 21 XP points now.

• Just two minutes ago he created another rideIntent. Thus he should have 26 XP points now.

C.1.5 BLOCK 5

concept CARPOOL {

point XP { name="Experience Point", abbreviation="XP", type=ADVANCING }

badge Epic { name = “EPIC”, image = "epic.png", hidden }

badge IceBreaker { name = "Ice Breaker", image="breaker.png", hidden }

level PennyPincher { name = "Penny Pincher", threshold=10 XP }

level NickelNurser { name = "Nickel Nurser", threshold = 15 XP }

level DonaldDuck { name ="Donald Duck", threshold = 20 XP }

mission Remind3Customers {

name = "Remind3Customers",

description = "Mission to 3 Remind 3 Customers",

init-rule

when player { has point XP, SUM > 3 }
then { give badge IceBreaker }

nextMissions { Remind10Customers }

}

mission Remind10Customers {

name = "The 10 Reminder Mission",

description = "Remind 10 Customers"

when player { has point XP, SUM >= 10 }
then { give badge Epic }

}
**GM01: WHAT DOES THE CODE DEFINE? [SINGLE-CHOICE]**

- Two missions, three levels, two badges, one point category.
- Two missions, four levels, two badges, one point category.
- Two missions, three levels, one badge, one point category.
- Four missions, three levels, two badges, one point category.

**GM02: MOST GAMIFICATION CONCEPTS FEATURE BADGES THAT USERS GET FOR SPECIAL ACTIONS. HOWEVER, SOME BADGES ARE SO HARD TO GET THAT IT WOULD NOT BE VERY MOTIVATING TO SHOW THEM TO THE USERS IN ADVANCE. WITHIN THE GIVEN CONCEPT, WHICH BADGE(S) IS/ARE KNOWN IN ADVANCE [MULTIPLE-CHOICE]**

- The Epic Badge
- The IceBreaker Badge.
- The Fail Badge.
- None of the three above.

**GM03: WHEN IS THE USER GIVEN THE ICEBREAKER BADGE? [SINGLE-CHOICE]**

- When the user has 3 or more XP points.
- When the user has 3 or more points in any point category.
- Never.
- It is given together with the EPIC Badge.

**GM04: ASSUME THAT ALICE HAS 12 XP POINTS. FILL THE TEXT WITH THE CORRECT INFORMATION. NOTE THAT FIELDS CAN REMAIN EMPTY IN SOME CASES [FREE-TEXT]**

- With the given XP points Alice should be the level called: Penny Pincher
- In addition, she should have the following badge(s): IceBreaker, Epic
- She has already completed the mission(s): Remind3Customers and Remind10-Customers
- She is entitled to do the following mission(s): Remind50Customer and YourNew-Job
C.1.6 BLOCK 6

concept CARPOOL {
  useraction A
  useraction B

  point XP { name="Experience Point", abbreviation="XP", type=ADVANCING }

  badge Epic { name = "EPIC", image = "epic.png" }

  badge IceBreaker { name = "IceBreaker", image = "icebreaker.png" }

  mission SomeNewMission {
    name="SomeNewMission",
    description="SomeNewMission"

    when player {
      ( 
        has badge IceBreaker 
        and did ua1 : useraction A 
        and did useraction B, this after ua1 
      ) or
      ( 
        is at location(lat=1.2, long=1.3) 
        and has luck P(0.7) 
      )
    )
    then {
      give badge Epic, give 10 XP
    }
  }
}

GC01: MARK ALL THE REWARDS FOR COMPLETING THE MISSION "SOMENEWMISSION". (MULTIPLE-CHOICE)

  • ... the Ice-Breaker badge
  • ... the EPIC Badge
  • ... 10 XP points.
  • ... between 3 and 7 XP points.

GC02: MARK ALL CORRECT CONSTELLATIONS THAT LEAD TO THE MISSION "SOMENEWMISSION" BEING FINISHED. (MULTIPLE-CHOICE)

  • ... have the IceBreaker Badge and trigger userevent A before userevent B.
  • ... have the Epic Badge and trigger userevent A before userevent B.
  • ... have the Epic Badge and trigger userevent B before userevent A.
  • ... have the IceBreaker Badge and trigger userevent B before userevent A.
• be in geo position 1.2, 1.3 and be lucky.
• be in geo position 1.2, 1.3 independent of any luck.
• trigger userevent A before B.
• trigger userevent B before A.

GC03: HOW WOULD YOU EXPECT THE LUCK(0.7) CODE TO WORK? (SINGLE-CHOICE)
• In 7 out of 10 cases the user is lucky.
• In 3 out of 10 cases the user is lucky.
• If the user already had luck 7 times, he will be unlucky next time.
• If the user already had luck 3 times, he will be unlucky next time.

• Bob.
• Trent.
• John.
• Alice.
• Mary.
### C.2 SYSTEM USABILITY SCALE

<table>
<thead>
<tr>
<th>#</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUS$_1$</td>
<td>I think that I would like to use this language frequently</td>
</tr>
<tr>
<td>SUS$_2$</td>
<td>I found the language unnecessarily complex</td>
</tr>
<tr>
<td>SUS$_3$</td>
<td>I thought the language was easy to use</td>
</tr>
<tr>
<td>SUS$_4$</td>
<td>I think that I would need the support of a technical person to be able to use this language</td>
</tr>
<tr>
<td>SUS$_5$</td>
<td>I found the various functions in this language were well integrated</td>
</tr>
<tr>
<td>SUS$_6$</td>
<td>I thought there was too much inconsistency in this language</td>
</tr>
<tr>
<td>SUS$_7$</td>
<td>I would imagine that most people would learn to use this language very quickly</td>
</tr>
<tr>
<td>SUS$_8$</td>
<td>I found the language very awkward to use</td>
</tr>
<tr>
<td>SUS$_9$</td>
<td>I felt very confident using the language</td>
</tr>
<tr>
<td>SUS$_{10}$</td>
<td>I needed to learn a lot of things before I could get going with this language</td>
</tr>
<tr>
<td>SUS$_{11}$</td>
<td>Overall, I would rate the user-friendliness of this language as:</td>
</tr>
</tbody>
</table>

Table C.1: System usability scale questions
C.3 CORRELATION MATRIX

![Correlation Matrix Diagram](image)

**Figure C.1: Correlation diagram for variables of GaML study**

---

1Significance levels: *** $< 0.1\%$, ** $< 1\%$, * $< 5\%$, $< 10\%$.  

---

233
D SUPPLEMENTARY PICTURES

D.1 DATA MODEL

Figure D.1: Class diagram for associations with image entity
D.2 GRAPHICAL ARTIFACTS OF GAMIFICATION PLATFORM

Figure D.2: Class diagram for event model

Figure D.3: Rule editor in the gamification platform prototype
D.2 Graphical Artifacts of Gamification Platform

Figure D.4: Gamification analytics in the gamification platform prototype

Figure D.5: Administration of user profile in the gamification platform prototype
Figure D.6: Additional widgets in the gamification platform prototype
Figure D.7: Optimized RETE graph representation for the SAP Networking Lunch application
The following section introduces important statistical tests and procedures which are used in this paper.

E.1 (MULTIVARIATE) ANALYSIS OF VARIANCE

Multivariate analysis of variance is used to show if the means of \( g \) samples with the same \( k \) features are significantly different. The null hypothesis is rejected when there exists a group mean that is significantly different from one other group mean. Analysis of variance requires two assumptions. First, multivariate normal distribution and homoscedasticity, that is, the variance of the group samples are unknown but equal [170, p. 423].

**HYPOTHESES:**

\[
\begin{align*}
H_0: \mu_1 &= \ldots = \mu_g \\
H_1: \exists \mu_i \neq \mu_j, \ j = 1...g
\end{align*}
\]

**TEST STATISTIC:**

It is assumed that there exists a data matrix \( X \) consisting of \( n \) cases (rows) and \( k \) features (columns) and \( \Lambda = \frac{\text{W}}{\text{T}} \), that is, the ratio of inter-group variance and intra-group variance. Hereby, \( T \) captures the variance that is shared between all groups and is calculated via \( T = \text{M}'\text{M} \) whereas \( M \) is the mean centered matrix for each feature \( M = \{m_{ij}\} = x_{ij} - \bar{x}_j, \ j = 1, ..., k \) with \( \bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij} \). Obviously, this can be also written using matrices operations: \( T = X'X - \bar{x}\bar{x}' = X'X - \frac{1}{n}(X1')(X1) \) [92]

Moreover, a matrix \( W \) is necessary that captures the variance within the groups. Given that \( g \) groups should examined, \( W = \sum_{i=1}^{g} W_i \). \( W_i \) is calculated in the same manner as \( T \) with the only difference that not all \( n \) cases are included but only cases that belong to this \( i \)th group.
Afterwards, a F-test can be applied to test if the variance shared between the groups is relatively larger than the variance shared within the groups. If yes, one can assume that different reasons are causing the means in the different groups.

\[ F = \frac{1 - \Lambda^\frac{1}{2} v_2}{\Lambda^\frac{1}{2} v_1} \sim F_{\nu_1,\nu_2,1-\alpha} \quad (E.1) \]

with \( v_1 = k(g - 1) \) and \( v_2 = s \left[ (n - 1) - \frac{k+g}{2} \right] - \frac{k(g-1)^2}{2} \) degrees of freedom whereas

\[ s = \sqrt{\frac{k(g-1)^2-4}{k^2+8(g-1)^2-5}} \] [92, p. 699].

Alternatively, a \( \chi^2 \) test exists

\[ \chi^2 = -[n - \frac{1}{2}(k + g)] ln \Lambda \sim \chi^2_{v,1-\alpha} \quad (E.2) \]

with \( v = k(g - 1) \) degrees of freedom [92, p. 699].

It is important to denote, that this is the most general case called multivariate analysis of variance (MANOVA), that is, for \( g \) groups and \( k \) features. Given that \( g = 2 \) and \( k = 1 \), the procedure is called analysis of variance (ANOVA) only and test statistics are much more reduced under this assumption [170, p. 424].

**TEST DECISION:**

The null hypothesis has to be rejected if either the F value or the \( \chi^2 \) value exceeds the self-chosen significance level \( \alpha \) of the respective distribution with \( \nu_1, \nu_2 \) or \( \nu \) degrees of freedom respectively.

**E.2 WILCOXON-MANN-WHITNEY TEST**

The Wilcoxon-Mann-Whitney Test (also called U-test) can be used if the distributional assumption of normality cannot be guaranteed across all \( k \) features. Thus, it is an alternative test statistic to (M)ANOVA. However, the test is designed for two samples only and not \( g \) groups as in (M)ANOVA. Furthermore, there exist only two assumptions for the U-test. First, both samples have to follow continuous distributions. Second, both samples have to be independent from each other [170, p. 390].

**HYPOTHESES:**

\[ H_0 : \mu_1 = \mu_2 \]
\[ H_1 : \mu_1 \neq \mu_2 \]

**TEST STATISTIC:**

The \( n \) observations of both samples are sorted in ascending order: \( x_1 \leq x_2 \leq \ldots \leq x_n \) whereas in this order one has to remember if the \( x_i \)th value is out of sample \( A \) or \( B \). Each value of the sorted sample gets a rank in accordance to its position. Afterwards a rank sum is calculated for the \( A \) and \( B \) samples, that is, \( R_A = \sum_{i=1}^{n} pos_{x_i \in A} \) and \( R_B = \sum_{i=1}^{n} pos_{x_i \in B} \).

Given that sample \( A \) consists of \( n_1 \) and sample \( B \) of \( n_2 \) cases, the \( U \) value is either

\[ U_1 = n_1n_2 + \frac{n_1(n_1 + 1)}{2} - R_A \quad (E.3) \]
E.3 Jarque-Bera Test

or

\[ U_2 = n_1n_2 + \frac{n_2(n_2 + 1)}{2} - R_B \]  \hspace{1cm} (E.4)

depending on whichever value is smaller [170, pp. 391-393].

**TEST DECISION:**

The null hypothesis is rejected if the \( U \) value is lesser equals than the critical value of a specific Wilcoxon distribution against a self-chosen significance level \( \alpha \):

\[ \hat{U} \leq U_{n_1,n_2,\alpha} \]  \hspace{1cm} (E.5)

Critical Values of the Wilcoxon distribution can be found in [140]

E.3 JARQUE-BERA TEST

The test of Jarque-Bera can be used as a statistical test to decide whether a sample is normal distributed or not. The null hypothesis declares, that the sample is normal distributed, the opposite hypothesis declares, that the sample is not normal distributed.

**HYPOTHESES:**

\[ H_0 : F(Y) = \Phi(Y) \]
\[ H_1 : F(Y) \neq \Phi(Y) \]

**TEST STATISTIC:**

The test statistic is based on kurtosis and skewness of the sample’s empirical distribution and is calculated as follows according to [102, S. 257].

\[ JB = T \left[ \frac{\mu_3^2}{6\mu_2^3} + \frac{1}{24} \left( \frac{\mu_4}{\mu_2^2} - 3 \right) \right]^2 \sim \chi^2(2) \]  \hspace{1cm} (E.6)

with \( \mu_j = \frac{1}{T} \sum_{t=1}^{T} u_{ij} = \frac{1}{T} \sum_{t=1}^{T} (x_t - \bar{x})^j \) representing the second, third or fourth moments of the normal distribution respectively.

**TEST DECISION:**

The null hypothesis \( H_0 \) is rejected, if the JB-value exceeds a self-chosen significance value of a \( \chi^2 \) distribution with two degrees of freedom.

E.4 T-TEST

The t-Test is used to decide whether an estimated parameter is significantly different from zero or not.

**HYPOTHESES:**

\[ H_0 : \beta_i = 0 \]
\[ H_1 : \beta_i \neq 0 \]
TEST STATISTIC:

The estimated parameter value is evaluated against the standard error of the parameter estimation. When estimated with maximum likelihood, the standard error itself is the standard deviation of the covariance matrix which is calculated from the inverse Fisher information matrix \( I(\beta_i) = -E(H)_{ii} \) whereas \( H \) is the Hesse matrix of all second order partial derivatives. Since maximum likelihood needs a distributional probability assumption, bootstrapping can be applied to calculate t-values when no distributional assumption can be made.

\[
t-value = \frac{\hat{\beta}_i}{se(\hat{\beta}_i)} = \frac{\hat{\beta}_i}{\sqrt{-E(H)^{-1}_{ii}}} \sim t_{T-1, (1-\alpha/2)} \tag{E.7}
\]

TEST DECISION:

The null hypothesis is rejected, if the t-Value is greater than the corresponding value of the student t-distribution with \( T-1 \) degrees of freedom with a self-chosen critical significance level. Since \( \beta_i \) is tested for equality one has to apply the two sided version of the test.
STATEMENT OF AUTHORSHIP

I declare that I have written this Dissertation, titled “Gamification as a Service - Conceptualization of a Generic Enterprise Gamification Platform” independently. There were no other references and resources used as stated in the work. I indicated literal or accordingly adopted quotations.

Dresden, July 9, 2014

Philipp Herzig, M.Sc.