GaML - A Modeling Language For Gamification

Philipp Herzig, Kay Jugel, Christof Momm, Michael Ameling
SAP AG
Chemnitzer Straße 48
01187 Dresden
firstname.lastname@sap.com

Alexander Schill
Technische Universität Dresden
Nöthnitzer Straße 46
01187 Dresden
alexander.schill@tu-dresden.com

***Abstract*—Gamification has become a trend over the last years, especially in non-game environments such as business systems. With the aim to increase the users' engagement and motivation, existing or new information systems are enriched with game design elements. Before the technical implementation, the gamification concept is created. However, creation of such concepts is an informal and error-prone process, i.e., the definition and exchange of game mechanics is done in natural language or using spreadsheets. This becomes especially relevant, if the gamification concept is handed over to the implementation phase in which IT-experts have to manually translate informal to formal concepts without having gamification expertise. In this paper, we describe a novel, declarative, and formal domain-specific language to define gamification concepts. Besides that the language is designed to be readable and partially writeable by gamification experts, the language is automatically compilable into gamification platforms without involving IT-experts.**

I. Introduction

Gamification, the *"use of game design elements in non-game contexts"* [1], has emerged as a trend for information systems (IS). Current studies provide empirical evidence that the addition of game design elements increases *participation* [2] or factors such as *flow* or *enjoyment* [3]. Therefore, the introduction of game design elements in IS, e.g. Enterprise Resource Planning, is a promising approach to increase the users' engagement on the job.

While gamification is considered as very promising in general, customers still face significant challenges when it comes to the implementation of concrete gamification concepts in business contexts.

First, a mechanism for the precise definition of gamification concepts is missing. Currently, gamification concepts are discussed in natural language transported through natural text, spreadsheets, or presentations (e.g., [4]). Since gamification is an inter-disciplinary method, we argue that this makes interchange and discussion of complex game mechanics between experts of different domains complicated, especially in business environments. 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. Ultimately, handing the concept over to an IT-expert for the implementation phase of the concept is an error-prone task. We refer to this problem as *P1*.

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., [5], [6]). However, all available platforms, except the one described in [7], [8], focus on very simple game mechanisms and are therefore not suitable for implementing sophisticated concepts. 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 [9] (*P2*).

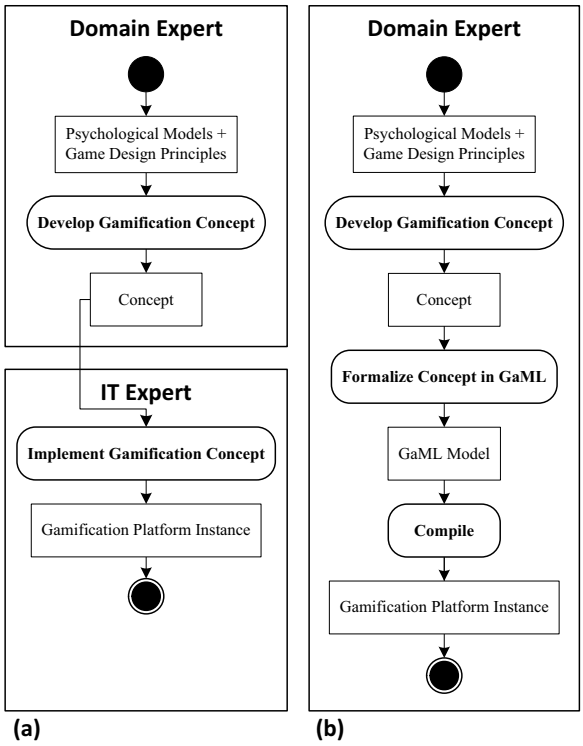

Fig. 1. Current vs. Proposed Approach

Third, even if the underlying technology may support complex gamification scenarios, the implementation becomes difficult as well since deep IT-knowledge is required [8].

CFP13UCC-USB/13 $26.00 © 2013 IEEE
DOI 10.1109/UCC.2013.96

Hence, it is not possible for domain experts anymore to precisely describe or maintain the gamification concept along its life-cycle (*P3*).

To address these issues, we describe a novel, declarative, and platform-independent gamification modeling language (*GaML*). *GaML* is designed in a way that all possible gamification concepts can be expressed by non-technical gamification experts. *GaML*-based models can then be automatically compiled to a specific technology without involving developers. Figure 1 illustrates the benefits of this approach by comparing the gamification development processes without *(a)* and with *(b) GaML*. While the To further demonstrate the value of the language, we will show its application within a productive car-sharing scenario where gamification has been introduced.

II. Requirements

The intended language should formalize the domain-specific concepts of gamification. Therefore, we present the used conceptualizations as requirements. Since researchers have not agreed on a common taxonomy of game concepts yet [10], we conducted a literature review to identify common elements. For structuring the requirements, we used a preliminary taxonomy provided by Deterding et al. [1]. In this taxonomy five levels for gamification design are introduced: *Game interface patterns* (L1), *Game Design Patterns* (L2), *Game Design Principles* (L3), *Game Models* (L4), and *Game Design Methods* (L5). While levels *L1* and *L2* are concerned with *what* visual concepts exist and *how* these elements relate to each other, all other levels (*L3-L5*) comprise design methods to create compelling game or gamification designs. The purpose of our proposed language is to support the formal definition of elements on *L1* and *L2* since they are directly related to the implementation of the gamification within an information system.

A. Basic Concepts (L1)

For the first level of Deterding's taxonomy, we describe the atomic visual elements of gamification in Table I. For each element we provide a name (i.e., the name used later in the language), possible synonyms, subtypes of the element, and references to its definition in literature. For example, we describe the visual element *Point* where possible synonyms are *Metric*, *Measure*, or *Currency*. Moreover, there are different types of points such as advancing, redeemable, karma, skill, or reputation points.

Further visual elements belonging to *L1* can be identified from the literature (Table II). However, 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, role, or skills for a particular player. A leaderboard is the aggregation of avatars and their points etc. All identified aggregations are presented in Table

B. Gamification Rules (L2)

Besides concepts on *L1*, we consider concepts on *L2* which concerns elements determining *how* the gamification is

Game Design Element	Synonyms	Subtypes	References
Point	Measure, Metric, Currency	Advancing, Redeemable Point, Karma, Reputation	[11], [12], [13]
Mission	Goal, Challenge	Individual, Collective	[14], [15], [16]
Roles	-	-	[17]
Skills	-	-	[17], [11]
Feedback	None	Informational: Points, Notification, Achievements, Narration; Corrective: Notifications	[16], [17], [18]
Event	User Actions	operative, resultant, external, interim	[14]
Achievement	Badge, Trophy, Virtual Good	expected, unexpected, partially (un-)expected	[11], [12], [19], [20]
Goods	-	Virtual, Real	[17], [12]
Narrative Context	Storytelling	-	[17], [16]
Notification	Alert	-	[14], [1], [16]

TABLE I
Visual (Basic) Game Mechanics

Game Design Element	Synonyms	Aggregates	References
Context	Objects' state	Points, Levels, Achievements, Goods, Skills	[14]
Avatar Levels	Level, Player Level	Points	[11], [12], [16]
Avatars	None	Context, Roles, Notifications, Avatar Levels, Goals	[11], [12], [16], [17]
Virtual Economy	Marketplace	Avatars, Virtual Goods, Virtual Currencies	[17]
Leaderboard	Highscore, Scoreboard	Points, Avatars	[11], [12]
Communication System	None	Notifications	[17]
Team	Cooperation	Avatars	[11], [18]

TABLE II
Visual (Aggregated) Game Mechanics

"played". According to [14], *"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* determine the existence of concepts on a meta-level, rules determine the gamification logic and how instances of these concepts evolve over time.

This evolution depends, for example, on:

- *User Actions* (e.g., user u does action a) (e.g., [14])
- *External Events* (e.g., mission completed during day or night) (e.g., [14])
- *Interim Events* (e.g., event as interim result) (e.g., [14])
- *Context* (e.g., user u has already 50 points) (e.g., [14]).
- *Constraints* such as temporal, spatial, Boolean, numeric, or random logic (e.g., user u did action a 1 day after completing mission b) ([13], [16], [14])
- *Randomness* (e.g., the user's action a is only considered

in 50% of all cases) ([14], [1], [16])
- *Joint Actions* (e.g., users u_1 and u_2 did action a together/against each other) ([11])
- or other user-independent or environment constraints (e.g., only first 50 users may get a badge).

These elements are considered as the conditional part of rules. Besides conditions, rules comprise consequences which are executed once the conditions become true. Further, consequences are the atomic elements of *L1* such as points, badges, or subsequent missions.

Thus, a formal language also has to take elements of *L2* into account. We acknowledge that the provided requirements may not be complete since very specific requirements may occur in some individual project. However, enhancing the language upon additional requirements is very simple as shown later.

III. Language Specification

In this section, we first describe the general design objectives of the language, followed by the language's syntax and semantics.

A. Design Objectives

We define four design objectives for the language in general. First, *GaML* formalizes the gamification concepts stated in the requirement section to address problem statement *P1*. A parser has to decide, whether an instance is well-formed according to the language's grammar or not. Second, a valid instance of *GaML* should be automatically compilable to gamification runtimes, e.g., gamification platforms (*P2*). Third, *GaML* should be at least readable for domain experts with minor IT knowledge, e.g., consultants or designers. Fourth, *GaML* should be fully writable for IT experts and partially writable by domain experts which addresses problem statement *P3*.

B. Model & Syntax

In this section, we describe excerpts of GaML's grammar. Additionally, we describe how the requirements map directly to the formal grammar.

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. We visualize an excerpt of *GaML*'s grammar using syntax railroads in Figure 2[1]. In this grammar, the definition of the model starts at S = *Model*, which produces the terminal *concept* followed by an *ID* and at least one type within curly brackets (Fig. 2a). Non-terminals N are visualized with gray boxes (e.g., ID or Type), terminals T in white boxes. The non-terminal *Type* can be any kind of game element (e.g., event, badge, and point) as shown in Figure *syntax*b). Hereby, each type represents one atomic element of *L1* concept of the requirements.

Each *L1* element is defined in more detail in the grammar using the productions. Figure 2 shows, for example, the definition of the element event (basic element according to

[1]The complete $LR(k)$ grammar in EBNF can be downloaded from http://gamification-inspired.com/GaML/

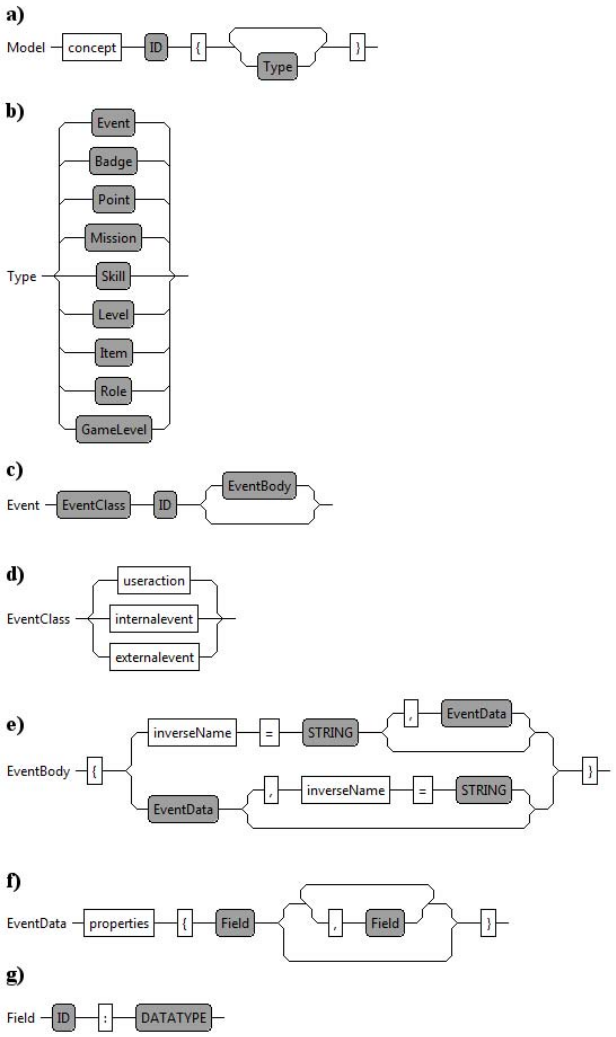

Fig. 2. Excerpt of GamL grammar

Table I). *Event* is started with either the keyword *useraction*, *externalevent*, or *internalevent*, followed by the event's name and an optional event body (Fig. 2c-d). Within the event's body the optional field *inverseName* and *data* can be defined (Fig. 2e). While *inverseName* refers to the name of the inverse event, *data* can be used to statically define attributes that describe the event further including their data types (Fig. 2f-g). We decided for static typing of attributes, since *GaML* is not supposed to be executed directly, but through intermediate languages (e.g., rule languages). Through the loss of transparency with this indirection, runtime type checking is omitted.

A very simple example of a well-formed *GaML* instance is presented in Listing 1. First, the instance is introduced with the keyword *concept* followed by the instance's name. Surrounded by curly brackets, the different types and elements of the instance are listed. There are two user actions *ride* and *rideIntent* of which the latter does also have some property definitions included which describe the event further. Additionally, the point

type *XP* (Experience Point) is defined. Besides basic data like the point's meta-type (i.e., ADVANCING) and abbreviation (i.e., XP) it also contains two conditions. Whenever the *when* part of the conditions is fulfilled, the *then* part is executed. In our example, the reception of an user action of type *rideIntent* which exists for at least 24 hours in the system, the player is given 1 XP. Further, when a *ride* event is recognized and the user (player) was the driver, he or she will receive 5 XP.

Note that in *GaML* not only event conditions are tested but all conditional aspects, such as temporal, spatial or contextual constraints as shown in Section II-B might be used. While adding conditions to a point directly allows the designer to model simple point rules, more complex conditions are considered as mission and, thus, declared within the *mission* element, which also allows for more complex consequences within the *then* block.

An important aspect of the syntax is that the *L1* game mechanics, such as events, points, badges or missions, are defined within the top level of the *GaML* instance, i.e., directly as antecedent of the global *concept*-block. *L2* game mechanics, however, are defined using conditions and consequences within *L1* types. We argue that this description of game mechanics leads to well-structured gamification models, which supports understandability and readability for domain experts.

```
concept CarPooling {

  useraction rideIntent

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

  point XP {
    name="Experience Point",
    abbreviation="XP",
    type=ADVANCING,

    when {
      useraction(type=rideIntent, lastsFor=24h)
    } then {
      give XP points: 1
    },

    when {
      useraction(type=ride, driver=true)
    } then {
      give XP points: 5
    }
  }
}
```

Listing 1. Well-formed *GaML* instance

C. Semantics

Besides syntax, we defined static semantics for the language, which are validated on the syntax tree of valid *GaML* instances. They are defined using logical expressions, e.g.,

$$\forall a, b : Badge(a) \wedge Badge(b) \wedge a \neq b \Rightarrow ID(a) \neq ID(b)$$

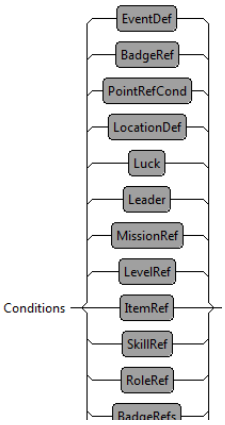

Fig. 3. Conditional elements in *GaML*

using object constraint language. In general, semantics are defined for the following aspects:

- Uniqueness of element IDs and attributes (e.g., all points must have unique names as their ID).
- Minimum and maximum cardinalities of children elements or properties (e.g., each *GaML* instance must have at least one event type defined).
- Cyclic dependencies in conditions and consequences (e.g., a mission should not be referenced by one of its child missions).
- 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., an advancing point must not be used as price metric for tradable items)

Some of these constraints may result either in errors (e.g., type checks) or warnings (e.g., point types). Dynamic semantics are not defined as the language cannot be executed at design time. However, it is possible to reduce dynamic semantics later on to the semantics of the target language of the specific gamification engine. Since we assume that a target language for execution exists, *GaML*'s semantics are intended to be described as compiler semantics, i.e., the semantics are translated through the compilation into a well-known language (e.g., Drools Rule Language, Prolog). Hence, neither operational nor axiomatic semantics are defined in *GaML*.

IV. Case Study

A. Scenario

To validate our language, we successfully modeled several real-world use cases from different domains (e.g., Finance, CallCenter, Sustainability) using *GaML*. In this paper, we present one of our live running use cases which covers the gamification of a carpooling system where people can share rides with each other. The solution is deployed in large company with around 8,000 regular users. In contrast to traditional commuting applications, users are automatically matched once

they posted their ride intents. However, the problem exists that people need to be motivated to share their car as a driver which is often perceived as loss of flexibility or convenience.

A qualitative analysis has been conducted to find out what the users motivate to use the application. 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, and perceive a smaller likelihood of accidents. Based on these observations, a gamification concept has been created to reinforce and uncover these intrinsic motivations.

This gamification concept is organized in several tasks and missions that the user has to fulfill. These tasks vary in difficulty, i.e., novice users get very simple tasks compared to experienced users who are faced by very complicated goals. For example, novice users have goals such as learning the user interface or create their first ride intent. Experienced users, however, are confronted with tasks such as active participation or network maintenance. To measure the user's progress and accomplishments, the gamification concept features experience points (XP) expressing the general experience and usage of the application, social points expressing the user's social network, numerous badges for special situations etc.

With regards to the other scenarios, we described the gamification concepts, for instance, for a HelpDesk application, a networking application, and a financial fact sheet mobile application. In these additional scenarios of course other types of point rules and conditions have been used. However, no additional language constructs were necessary as the ones described in our current grammar. Hence, we can show that our concept can be applied across multiple use cases and scenarios. For the sake of brevity, however, we only present the car-sharing scenario below.

B. Implementation

We developed *GaML* on top of our existing gamification platform which has been described in [7], [8]. Moreover, this particular platform runs productively as gamification service for the car-sharing scenario.

For the language, we developed an editor[2] including lexer and parser using xText. With the editor, it is possible to check instances of *GaML* with regards to syntax and static semantics. Moreover, the editor provides visual highlighting of the language its keywords or terminals and provides code completion for convenient writing. A small excerpt of the scenario is presented in Figure 4.

For example, the point type *XP* comprises three conditions. In each condition, points are given to the user issuing these events (lines 22-34). It is also possible to calculate the amount of points using numeric expressions with references to event properties (line 33). Finally, even complex operations such as joins across users are possible as presented in lines 48 *ff.* This expression can be interpreted as: if a combination of two users occurs who shared a ride together but never shared a

[2]The latest version of the GaML editor can be downloaded from: http://gamification-inspired.com/GaML/GaMLEditor.zip

CarPool.gaml

```
1  concept CARPOOL {
2
3      useraction rideIntent
4      useraction ride {
5          properties { carbonImpact:Decimal,  kilometers:Decimal,
6              driver:Boolean, rideId:Number
7          }
8      }
9
10     internalevent socializerEvent {
11         properties {pid:Text, friend:Text}
12     }
13
14     point Socializer {
15         name="Socializer Point", abbreviation="SP"
16     }
17
18     point XP {
19         name="Experience Point", abbreviation="XP",
20         type=ADVANCING,
21
22         when {
23             useraction(type=rideIntent, lastsFor=24h)
24         }
25         then {
26             give points: 1
27         },
28
29         when {
30             evt1:useraction(type=ride, driver=false)
31         }
32         then {
33             give points: 10+evt1.carbonImpact/10
34         }
35     }
36
37     level Rookie {
38         name = "TwoGo Rookie", threshold = 5 XP
39     }
40
41     level TricycleRider {
42         name = "TwoGo Rookie", threshold = 10 XP
43     }
44
45     mission GetSocial {
46         when {
47             evt1:useraction(type=ride)
48             evt2:useraction(type=ride, rideId=evt1.rideId, player<>evt1.player)
49             internalevent(type=socializerEvent, pid=evt1.player, friend=evt2.player)
50         } then {
51             create event socializerEvent(pid=evt1.player, friend=evt2.player)
52         }
53         when {
54             internalevent(type=socializerEvent)
55         } then {
56             give Socializer points: 1
57         }
58     }
59
60     point Carbon
61         name="Car
62         abbreviat
63
64         when { ev
```

Carbon - CARPOOL.Carbon
Kilometer - CARPOOL.Kilometer
Socializer - CARPOOL.Socializer
XP - CARPOOL.XP
badge:
points:

Fig. 4. Carpooling example in *GaML* Editor

ride before, an interim *socializer* event is created which, in turn, gives one *socializer* point to each passenger.

Although the language was designed to be partially writeable by domain experts, we acknowledge that, especially the latter example, is not writable for this target group anymore. Hence, we argue that additional expertise, e.g., from an IT expert is necessary to formulate more complicated game mechanics as the one shown above. Nonetheless, we argue that *GaML* might be well suited to allow conversation across IT and domain experts and, thus, makes the process of formalizing the gamification concept for the implementation less error-prone.

V. Related Work

The state of the art in the field of gamification currently lacks a proper modeling language for gamification concepts, which makes this work even more relevant. However, there are other languages or modeling approaches, but none of them suffices for modeling a complete gamification concept.

In [21] the game description language (GDL) is described as language for modeling common games, e.g., card or dice games as well as simple games like *tic-tac-toe*. Although very powerful and expressive, the syntax is very technical and based on predicate calculus, which makes it arguably hard for non-technical experts to read or write instances. The goal of this language is to provide a formal definition of games so that computers may automatically learn and play the game against others (computers or humans) just by the formal description. Hence, the design goals completely differ from our assumptions.

The Agent Modeling language (AML) proposed in [22] discusses an entirely different approach and tries to model agents' behavior, such as the behavior of a player, from a designer's point of view. It allows for modeling the same aspects as the GDL in a more non-expert readable way, but still suffers from being limited to normal games only. It is thus not usable for modeling complex gamification concepts.

The approach described in [23] presents a card game modeling language, which can model the different rules, cards and stages along with other entities used in card games. The focus lies not on only on playing and describing the games but modifying them to create new games. However, the syntax has been explicitly designed for IT-experts and is, thus, hardly readable and writable for domain experts. Moreover, it is not possible to model gamification concepts as stated in the requirements.

There are more research approaches for gamification as well as games, however, none of these tackle the problem of creating formal gamification models in a generic way which is human readable on the one hand and provides a precise and unambiguous description of gamification concepts on the other hand.

VI. Summary & Outlook

In this paper we proposed *GaML*, a language for modeling gamification concepts in formal, rigorous way. The primary design goal for this language is, that it is at readable by domain experts such as consultants or designers with minor IT background. Moreover, the language serves as a vehicle to decouple the conception phase from the implementation phase and, thus, allows for the definition of technology-independent gamification concepts. Finally, the language allows the precise discussion, comparison, and validation of game mechanics or gamification concepts between researchers and practitioners.

In future work, we implement the full compilation from *GaML* into an executable gamification concept within target gamification platforms, e.g., SAP Gamification Platform. This constructively proofs that *GaML* is an executable language later. Additionally, the full set of static and compiler semantics are presented. Finally, we are going to evaluate qualitative properties of the language such as readability and writability for domain experts or IT-experts respectively.

References

[1] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From Game Design Elements to Gamefulness: Defining Gamification," in *MindTrek '11 Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. ACM, 2011, pp. 9–15.

[2] J. Thom, D. R. Millen, and J. DiMicco, "Removing Gamification from an Enterprise," in *Proceedings CSCW*, 2012, pp. 1067–1070.

[3] P. Herzig, S. Strahringer, and M. Ameling, "Gamification of ERP Systems - Exploring Gamification Effects on User Acceptance Constructs," in *Multikonferenz Wirtschaftsinformatik*. GITO, 2012, pp. 793–804.

[4] R. S. Brewer, G. E. Lee, Y. Xu, C. Desiato, M. Katchuck, and P. M. Johnson, "Lights off. game on. the kukui cup: A dorm energy competition," in *Proceedings of the CHI 2011 Workshop Gamification: Using Game Design Elements in Non-Game Contexts*, 2011.

[5] Bunchball Inc., available on http://www.bunchball.com/ (retrieved on 09.05.2012).

[6] Badgeville, The Behavior Plattform, available on http://www.badgeville.com/platform/ (retrieved on 09.05.2012).

[7] P. Herzig, M. Ameling, and A. Schill, "A Generic Platform for Enterprise Gamification," in *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*. IEEE, 2012, pp. 219–223.

[8] P. Herzig, B. Wolf, S. Brunstein, and A. Schill, "Efficient Persistency Management in Complex Event Processing: A Hybrid Approach for Gamification Systems," in *Theory, Practice, and Applications of Rules on the Web*, ser. Lecture Notes in Computer Science, L. Morgenstern, P. Stefaneas, F. Lvy, A. Wyner, and A. Paschke, Eds. Springer Berlin Heidelberg, 2013, vol. 8035, pp. 129–143. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39617-5_14

[9] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA - Service-oriented Architecture Best Practices*. Prentice Hall International, 2005.

[10] Björk, Staffan and Holopainen, Jussi, *Patterns in Game Design*. Charles River Media Inc., 2005.

[11] G. Zichermann and C. Cunningham, *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. O'Reilly Media, 2011.

[12] Bunchball Inc., "Gamification 101: An Introduction to the Use of Game Dynamics to Influence Behavior," 2010, available on http://www.bunchball.com/gamification/gamification101.pdf (retrieved on 27.07.2011).

[13] Kevin Werbach and Dan Hunter, *For the Win: How Game Thinking Can Revolutionize Your Business*. Wharton Digital Press, 2012.

[14] J. Schell, *The Art of Game Design: A Book of Lenses*. Elsevier Inc., 2008.

[15] J. P. Zagal, M. Mateas, C. Fernández-Vara, B. Hochhalter, and N. Lichti, "Towards an Ontological Language for Game Analysis," in *DiGRA 2005 Conference*, 2005.

[16] K. M. Kapp, *The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education*, R. Taff, Ed. Pfeiffer, 2012.

[17] B. Reeves and J. L. Read, *Total Engagement: Using Games and Virtual Worlds to Change the Way People Work and Businesses Compete*. Boston, MA: Harvard Business Press, 2009.

[18] J. McGonigal, *Reality is Broken: Why Games Make Us Better and How They Can Change The World*. New York: The Penguin Press, 2011.

[19] M. Montola, T. Nummenmaa, A. Lucerano, M. Boberg, and H. Korhonen, "Applying Game Achievements Systems to Enhance User Experience in a Photo Sharing Service," in *MindTrek'09: Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, 2009, pp. 94–97.

[20] J. Hamari and V. Eranti, "Framework for Designing and Evaluating Game Achievements," in *Proceedings of DiGRA 2011 Conference*, 2011.

[21] M. Thielscher, "A general game description language for incomplete information games." in *AAAI*, vol. 10, 2010, pp. 994–999.

[22] R. Červenka, I. Trenčanskỳ, M. Calisti, and D. Greenwood, "Aml: Agent modeling language toward industry-grade agent-based modeling," in *Agent-Oriented Software Engineering V*. Springer, 2005, pp. 31–46.

[23] J. M. Font, T. Mahlmann, D. Manrique, and J. Togelius, "A card game description language," in *Applications of Evolutionary Computation*. Springer, 2013, pp. 254–263.