Engineering Gameful Applications with MPS

Antonio Bucchiarone, Antonio Cicchetti, and Annapaola Marconi

Abstract Gamification refers to approaches that apply gaming elements and mechanics into contexts where gaming is not the main business purpose. Gamification principles have proven to be very effective in motivating target users in keeping their engagement within everyday challenges, including dedication to education, use of public transportation, adoption of healthy habits, and so forth. The spread of gameful applications and the consequent growth of the user base are making their design and development complexity to increase, e.g., due to the need of more and more customized solutions. In this respect, current state-of-the-art development approaches are either too close to programming or completely prepackaged. In the former case, domain and gamification experts are confronted with the abstraction gap between the concepts they would like to use and the corresponding implementation through coding. In the latter situation, customization opportunities are remarkably limited or require again hand-tuning through coding. In both scenarios, programmer tasks are tedious and error-prone, given the intrinsic characteristics of gamified applications, which are sets of rules to be triggered as a consequence of specific events.

This chapter illustrates the language engineering endeavor devoted to the creation of the Gamification Design Framework (GDF) through MPS. GDF is conceived by pursuing two main principles: correctness-by-construction and automation. The former aims at providing a language infrastructure conveying consistency between the different aspects of a gameful application in an intrinsic way. The latter aspires to maximize generative features in order to reduce coding needs. As a result, GDF is implemented by means of MPS as a set of three-layered domain-specific languages, where a lower-level language instantiates and extends the concepts defined from the language(s) above. Moreover, GDF is equipped with generators to automatically

A. Bucchiarone (✉) · A. Marconi
Fondazione Bruno Kessler (FBK) - MoDiS, Trento, Italy
e-mail: bucchiarone@fbk.eu; marconi@fbk.eu

A. Cicchetti
School of Innovation, Design and Engineering (IDT), Västerås, Sweden
e-mail: antonio.cicchetti@mdh.se

A. Bucchiarone et al. (eds.), *Domain-Specific Languages in Practice*,
https://doi.org/10.1007/978-3-030-73758-0_8

create gameful application structural components, behaviors, and deployment into a selected gamification engine.

1 Introduction

Playing is an activity humans do since their birth for (self-)learning, to meet others, to be part of communities, to relax, and so forth. Indeed, we are so used to it that we spend a growing amount of our free time with some form of gaming even in adulthood, and people made a profession out of it [1]. Interestingly, psychologists also observed that by introducing gaming elements and mechanics into "normal"—non-gaming—tasks, it can be possible to promote engagement and even motivate people to achieve certain objectives. As a matter of fact, an increasing number of activities include gamification elements, very often supported by software applications: Internet banking, sport/activity trackers, and shopping/traveling fidelity cards are all (few) examples of application domains targeted by gamification [2].

A fundamental concern of gameful applications[1] is their tailoring to the target domain and users: if a game is detached from the domain interests, the risk is to promote counterproductive/undesired behaviors; similarly, too easy or too complex games could fail engagement objectives due to loss of interest or discouragement, respectively [3]. A direct consequence of the mentioned tailoring needs is the critical contribution and cooperation of application domain and gamification experts: the former ones provide inputs about the engagement issues and desired outcomes, while the latter ones propose corresponding gamification strategies. Such a cooperation conveys gameful application specifications to be implemented in an appropriate target platform.

In the current state of practice, one available implementation option is to pick up a prepackaged gamification application from a repository [4]. The advantage would be to have a quick development phase limited to configuration purposes, at the price of very limited customization possibilities, unless manually tuning the existing implementation. Diametrically opposite, a completely new gamified application can be developed from scratch: this solution necessarily entails longer time to market, with the advantage of realizing a fully customized implementation. Regardless of the choice, the realization and deployment phases introduce an abstraction gap between gamification stakeholders, namely, domain and gamification experts, and the gameful application itself. In fact, the target application is typically implemented as a collection of rules matching incoming event notifications with corresponding game status updates. Therefore, developers need to translate game mechanics and other elements into corresponding rules, while the other stakeholders are required to backtrack state changes into corresponding gaming events.

[1] In the remainder of this chapter, we will always refer to software supported gamification, unless explicitly mentioned.

With the growing adoption of gamification in disparate application domains and its spread to a wider range of users, the complexity of gameful software is unavoidably increasing. In this respect, the abstraction gap between design and realization becomes a critical issue: the implementation phase is more tedious and error-prone, due to the number of rules and the customization needs. Moreover, maintenance and evolution activities are harder to manage, due to the disconnection between design and realization.

In order to close the gap between design and implementation of gameful applications, we proposed the Gamification Design Framework (GDF) [5, 6]. GDF is a collection of domain-specific languages (DSLs) devoted to the specification, implementation, and deployment of gameful applications. The framework has been developed by the following three key principles:

Separation-of-concerns : a gamification approach can be described by means of several perspectives. When the complexity grows, an effective way to alleviate it is to manage different perspectives as separate points of view that are later on fused into a complete solution;

Correctness-by-construction : given the growth of gamification employment and range of its potential users, the specification of gameful applications becomes increasingly intricate. In this respect, game rules shall be consistent with mechanisms and elements intended for the target application;

Automation : in order to close the gap between design and implementation, the amount of manually written code shall be reduced as much as possible. Or in the other way around, the degree of automation provided by the framework shall be maximized.

GDF actualizes the mentioned key principles by means of three DSLs that correspond to three abstraction layers any gamified application can be viewed through: (i) the topmost layer defines general mechanics and elements a solution could include, e.g., the concept of point, bonus, challenge, etc.; (ii) the second layer instantiates a subset of the abstract concepts defined on the level above due to the specification of the gameful application under development, for example, number of steps, walker of the week, hundred thousand steps week, respectively; (iii) the third and bottom layer describes the implementation of the concepts above together with their deployment on a gamification engine. Here, configuration parameters can be set, like thresholds to gain points, bonuses, and awards, the timing of challenges activation, and the assignment of players and teams to the defined tasks. Moreover, the layers convey generators enabling the automated derivation of implementation code for the gamified application.

GDF is practically realized by means of Jetbrains MPS and is the result of a challenging language engineering process. In particular, the DSLs included in GDF required a language workbench enabling meta-modeling, semantics specification through generators, and multi-view-based modeling support to ensure the consistency between the different points of view. During the language engineering process, we soon faced the problem of modeling a system of constraints (as it is a gameful application), which tended to be intractable by adopting diagrammatic approaches.

Moreover, we needed a mechanism enabling the introduction and refinement of high-level gamification solutions and concepts without requiring domain experts to modify the language specification, e.g., by adding new consistency checks. As a consequence, the DSLs included in GDF convey a text-based concrete syntax that eases the definition of game rules. Moreover, they exploit the language extension mechanisms provided by MPS to define the interconnections between the different abstraction layers, which implicitly ensures consistency through inheritance relationships.

The remainder of the chapter is structured as follows: Sect. 2 discusses in detail the motivation and contributions of our work. Then, Sect. 3 introduces the case study used to validate GDF. Section 4 illustrates the main components and features of GDF and the included DSLs. Eventually, Sect. 5 discusses advantages, drawbacks, and open challenges in the use of MPS for implementing GDF, including possible future investigation directions, while Sect. 6 draws conclusions about the chapter.

2 Motivations and Contribution

Gamification is a relatively recent field of research consisting of developing game characteristics in non-game contexts [2, 7]. It is a method that focuses on triggering elements of human behavior and psychology in order to provide rewarding, engaging, and exciting experiences for the users. While most of the elements that gamification uses are borrowed from the games (points, badges, leaderboards, etc.), it is specifically intended to help boost engagement and make products more appealing to the user. Gamification concepts and their usage have been reviewed in several research works with positive feedback, yet some doubts persist about its effectiveness. Nevertheless, over the last years, various research studies focused on introducing gamification in software engineering environments [8].

Gamification has shown a wide range of possible uses, varying from gaming in education [9] to online marketing and software apps [10, 11]. Bartel and Hagel presented a learning concept based on a gamification approach to promote students' motivation and engagement in their university education [12]. Moreover, Toda et al. developed an approach for planning and deploying gamification concepts by means of social networks within educational contexts [13].

While many studies have theorized about game design, the two most cited frameworks are **MDA (Mechanics, Dynamics, and Aesthetics)** by Hunicke et al. [14] and the **Elemental Tetrad** by Schell [15]. Rodrigues et al. [16] have conducted a review study that analyzed the literature covering 50 papers published over the time period of 2011 to 2016 using Leximancer software. The study determined and shaped the main concepts proposed in gamification researches. Thirty concepts have been identified related to the gamification main domains. These concepts can be classified by level of relevancy and fit under the fundamental components of gamification frameworks.

Figure 1 includes the two most adopted gamification frameworks, being MDA and Element Tetrad, detailing their concepts and the elements that are included in each concept as reported in the following sections. The figure shows the connectivity between these two frameworks and demonstrates that although they are two different frameworks, they are built on many common basics and pillars.

2.1 MDA

Mechanics, Dynamics, and Aesthetics framework or simply known as MDA facilitates a deliberation of differences between designer and player perspectives. In other words, from the designer's perspective, mechanics produce dynamics, which then produces aesthetics, while, according to a user's perspective, MDA converts into rules, system, and fun. MDA provides one approach of understanding games and how gamification works [17–19].

The elements of MDA are defined as follows:

- **Mechanics.** The concept of mechanics describes the particular components of the game, at the level of data representation and algorithms [14]. Game mechanics involve the distinct set of rules that dictate the outcome of interactions within the system. Points, badges, leaderboards, statuses, levels, quests, countdowns, tasks/quest/missions, and other particular rules and rewards all fall under the category of game mechanics [18]. Moreover, these elements fall under three groups which are the components, the controls, and the courses. Three different types of mechanics are extremely important in games and in gamified experiences: setup mechanics, rule mechanics, and progression mechanics[19].
- **Dynamics.** Hunicke describes dynamics as the runtime behavior of the mechanics acting on the players' inputs to the game and the results of this player interaction in the game over time [14]. Game dynamics refer to the principles that create and support aesthetic experience. Unlike the game mechanics set by the designer, game dynamics describe in-game behaviors and strategic actions and interactions that emerge during play [20], such as context, behavior, consequences, and achievements. Examples of game dynamics contain a sequence of chance, constraints, behaviors, consequence, and finally the achievement.
- **Aesthetic.** It describes the desirable emotional responses evoked in the players, when they interact with the game system [14], in other words, how the game does look, feel, and sound. Aesthetics encompass the various emotional goals of the game: sensation, fantasy, narrative, challenge, fellowship, discovery, expression, and submission [18]. Therefore, aesthetics are the result of how players follow the mechanics and then generate the dynamics. Aesthetics is what gives appeal and fun to the users. Assuming that players will stop playing a game if they do not enjoy themselves, then creating player enjoyment should be the main goal [19].

2.2 Elemental Tetrad

Tetrad proposes that a gamification comprises four concepts being aesthetics, mechanics, story, and technology [15]. None of the four elements have higher importance than the other. However, technology tends to be the least visible to the user, while the aesthetic is the most visible. The concepts of this framework are as follows:

- **Technology.** Technology refers to the tools and systems used to implement or deliver the gameplay. Everything including coding, software, and the devices the user has in their hands falls under technology. Any input or output or choice of technological has an impact on the design of the game.
- **Mechanics.** Schell defines mechanics as the procedures and rules of a game and discusses six mechanics—space, objects, actions, rules, skill, and chance. Space is where the users engage with the game (both virtual worlds and physical space). Objects are tools used by the player to advance in the game. Actions are how the player interacts with objects. Rules govern the game environment. Skills are physical, mental, and social abilities used by a player to progress. Chance refers to the randomness and uncertainty that exist in games [15]. Some inconsistency is shown here as space, objects, actions, skill, and chance are not procedures and rules.
- **Aesthetics.** In most of the concepts that employ it, describes how the game looks, sounds, smells, tastes, and feels. Aesthetic is what the players are most familiar with as it represents everything they see, hear, feel, and perhaps in certain situations even taste and smell during the experience of the game. This is where the game connects to the players' senses. Every piece of art and sound is part of this element.
- **Story.** This element contains the journey, the incredible worlds players find themselves in during gameplay, and the personal relationships between all of the characters in a game. Besides that it can also contain educational or other content that needs to find its way into the game. A way to describe story and differentiate the perspectives of end users and designers is through narratives which can be sorted in three types:
  - *Embedded narrative.* It represents the view of the game designer in terms of structured components and event sequences intentionally embedded in a system by the designers. Hence, embedded narratives align conceptually with game mechanics.
  - *Emergent narrative.* It is created by players during their interaction with the gamification application in a dynamic fashion as they perform different activities. In this way, emergent narratives correspond conceptually to *game dynamics*.

– *Interpreted narratives.* It characterizes the end user's ascribed meaningfulness of experiences with the gamification activities. Given that these narratives are mental representations of the players, they are logically aligned with the concept of game aesthetics.

2.3 Open Issues and Contribution

Although the Mechanics, Dynamics, and Aesthetics (MDA) framework is probably the most widely accepted and practically employed approach to game design, MDA framework has recently been criticized for several weaknesses [2] Other frameworks have been proposed to overcome those limitations, but none has generated sufficient support to replace MDA.

Few studies utilized a more nuanced conceptualization of game narratives and suggested a combination of MDA and the Elemental Tetrad. Ralf and Monu[3] referred to their concept as the Mechanics, Technology, Dynamics, Aesthetics plus Narratives Framework (MTDA+N). This framework, despite its limitations, can serve as a useful theorized concept for teaching fundamentals of game design and clarifying some core concepts especially the game narratives.

Walk et al. [21] aimed to overcome the weaknesses of the well-established MDA framework by placing it on new pillars. They presented the Design, Dynamics, Experience (DDE) framework for the design of computer and video games. Their framework is based on what they determined to be what needs to be produced during the design and development of the game. In addition, they highlight the role of this asset in its contribution to the game experience [21].

A useful and communicable theory of game design is needed to help game designers and academics speak a common language, to legitimize the study of game design among other social sciences, and to educate the next generation of game designers. While game design is a discipline with its own rules, gamification is more of a method that facilitates engagement and entertainment. When working on game design and/or integrating gamification into a product, it is important to keep in mind the users' motivations, feelings, and contexts. The application of **model-driven engineering for gamification** has been studied over the past years and showed promises. However, the drawback is that the applications have been particular to specific cases which limits the interest as the approach of each meta-model would vary from case to case.

[2] https://gamedesignadvance.com/?p=2995.

[3] http://www.firstpersonscholar.com/a-working-theory-of-game-design/.

One important issue hindering the adoption of gamified solutions is that currently they are conceived as separate mechanisms to be designed and developed as side applications. This means that, for example, a teacher willing to introduce gaming aspects to her course is required to create her own gameful application, possibly customized for course content and learning objectives, or alternatively enrich an existing learning platform with gamification elements. Even more, in both cases, the specification of the target application would require expertise in game development and programming skills since there exists a gap between the definition of games and their concrete implementation and deployment. In this respect, we envisioned the definition of **a software engineering approach for gameful applications** such that the *game* part should be designed in its main ingredients and deployed on an appropriate *gamification engine* [22]. Based on this, gameful concepts and elements would be handled as specific concerns of software applications. Therefore, the gaming aspects would be kept separate and plugged in existing applications instead of requiring ad hoc extensions of the applications themselves. The main advantages of this vision are increased scalability and maintainability of the gameful mechanisms, which would not get intertwined with irrelevant aspects from a gamification point of view. Moreover, it would make it easier to integrate gamification elements into existing applications.

As a step forward the realization of our vision, we proposed a **Gamification Design Framework (GDF)** [5, 6]. GDF is a solution for the design and deployment of gamified applications through model-driven engineering mechanisms. In particular, it is based on a set of well-defined modeling layers that start from the definition of the main gamification elements, followed by the specification on how those elements are composed to design games, and then progressively refined to reach concrete game implementation and execution. The layers are interconnected through specialization/generalization relationships such that a multilevel modeling approach is realized [6]. The choice of a multilevel modeling approach came out directly from the nature of gamification applications: gamification principles, instantiated in terms of game elements, in turn materialized as game element instances [23].

To characterize GDF and to make it conform to the frameworks already introduced, such as those presented in Fig. 1, we tried to define it incrementally covering for now some of the concepts and the elements already introduced in the literature. Our goal is not to introduce a new conceptual framework for gamification but a modular, extensible, and useful environment to program gamified applications in different contexts and with different objectives. Figure 2 shows in green the concepts and the elements supported now by GDF and presented in this chapter.

The language engineering process toward the concrete implementation of GDF required a relevant effort due to the intrinsic characteristics of the problem domain (i.e., gamification), the need for allowing all the stakeholders to participate to game specifications, and the availability of adequate language workbenches (see a deeper discussion in Sect. 5). By going into more details, gamified applications are mainly

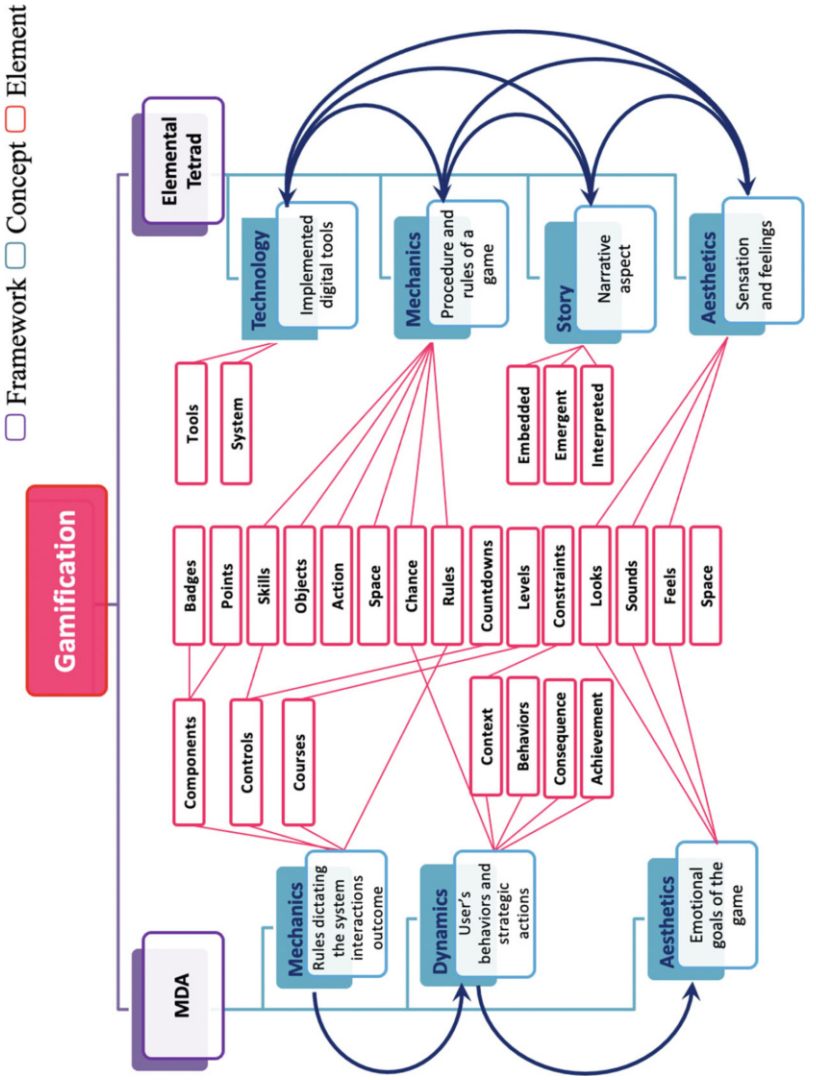

Fig. 1 Gamification frameworks and their concepts

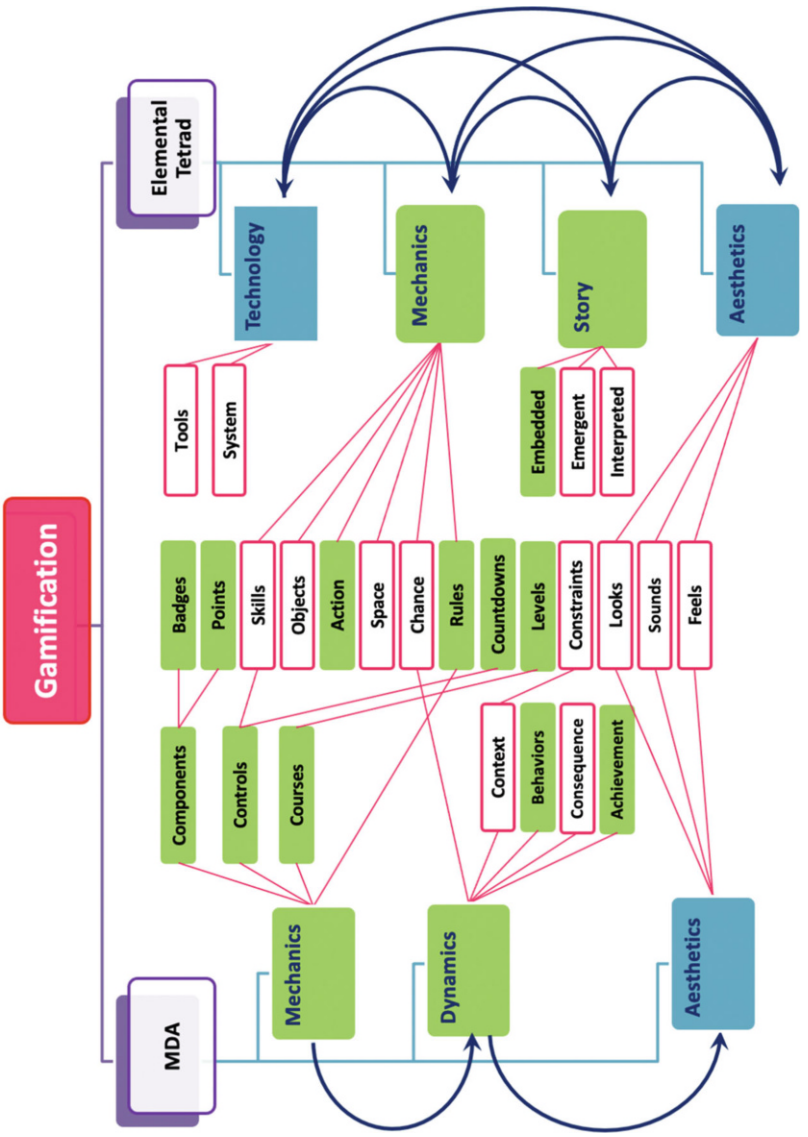

Fig. 2 Gamification concepts and elements supported by GDF

collections of rules orchestrating challenges, players'/teams' actions, points and bonuses gains, and so forth. In this respect, we need a solution able (1) to be easily used by those defining new gamified applications and (2) to be easily extended when switching from one application domain to another. As a consequence, we needed a language workbench enabling easy maintenance/extension of the DSLs and also relieving the users of consistency management.

After several attempts and failures with different alternatives, we opted for MPS workbench. The details of GDF realization by means of MPS are carefully described in Sect. 4 where we present how we have exploited the main features offered by this workbench, notably the projectional editors, the language extension mechanisms, and the support for code generators.

3 Case Study: PapyGame Design with GDF

We have used and validated GDF in diverse application domains, like smart mobility and education [5]. In this section, we report further details about another recent usage scenario, that is, how we have used GDF as core component of a specific gamified software modeling environment called **PapyGame**[4] The choice of this case is due to its consistency with our vision of future software engineering of gamified applications (see Sect. 2.3) and the required contribution of multiple stakeholders. In particular, in PapyGame, the Papyrus[5] modeling tool has been extended with gamification features. The objective of the gamification is to help master degree students in Computer Science in learning specific modeling aspects using Papyrus for UML [24].

Each *student assignment* in PapyGame is composed of a set of *levels* (grouped in series) that each student should deal with. For each level, an exercise is assigned, and each passed level unlocks the next exercise of the next level. To start a PapyGame session, the player must first enter their login and password. Once connected, PapyGame displays a dashboard (see Fig. 3) representing the series of the player. Each completed (successfully) level is displayed in green with the corresponding number of gold coins (GC) and the experience points (XP) rewarded. Remaining levels are colored in gray with a lock except for the first one which is the next level to be played (unlocked).

Each exercise is associated with a specific game type: the *Hangman*, *when a new part of the man drawing is added with every wrong answer*, and the *On Your Own (OYO)*, *when the student executes the exercise with no help*. At the same time, each

[4] https://www.papygame.com.

[5] https://www.eclipse.org/papyrus/.

Fig. 3 PapyGame Dashboard UI

Level 1
Type: OYO
Goal: Learn to build Classes
Description: Let's imagine we want to design a database to catalogue all our books.
To begin with, we want to represent the concepts of book but also its authors and publishing company. Please create following *UML Classes*.
For *class names*, use the *Java class naming convention* (Word1Word2Word3).

Rewards

Errors	Time	XP	GC
0	--	50	2
1	--	30	1
2	--	20	0
3+	--	10	0
Time Rewards			
XP+=	50 – time/2		
GC+=	1 if time < 60		

Correct diagram

Fig. 4 PapyGame: On Your Own (OYO) example

exercise has an associated set of point concepts (experience points, gold coins, etc.) and rules. All these aspects are defined by the teacher using GDF. In particular, the teachers execute the following steps in defining each level: (1) choose a game type among the available games in the system (i.e., Hangman, OYO), (2) define the goal and the description of the level, (3) create the expected (correct) diagram in Papyrus according to the level objective, and (4) create the reward rules about points and eventual bonuses. Figures 4 and 5 show two examples of PapyGame levels expressed in natural language format in a teacher document.

GDF is used to make this design task of the teacher automatically saved and deployed in the PapyGame backend. This provides a way to define all the game elements that regulate the game behavior through specific modeling editors. In particular this is possible exploiting the editors provided by GDF and its related generators (see details in Sect. 4). Once the *student assignments* are designed and deployed, the students can select the respective levels and start to play and

Level 2
Type: Hangman
Goal: Learn to build Properties with primitive Types
Description: Papyrus imports in each UML model a *PrimitiveTypes package* that contains the String, Integer or Boolean types. Use these types to define *attributes in the classes* of the previous level. A book is characterized by a title (text), an edition year (integer) and a format (text). An author is characterized by a last name (text) and a first name (text). A production company has a name (text) and a (text). Please use the *Java naming convention*, but for attributes this time (word1Word2Word3). For *class names*, use the Java class naming convention (Word1Word2Word3).

Rewards

Errors	Time	XP	GC
0	--	70	1
1	--	55	0
2	--	40	0
3+	--	25	0
Time Rewards			
XP+=	70 – time		
GC+=	1 if time < 30		

Correct diagram

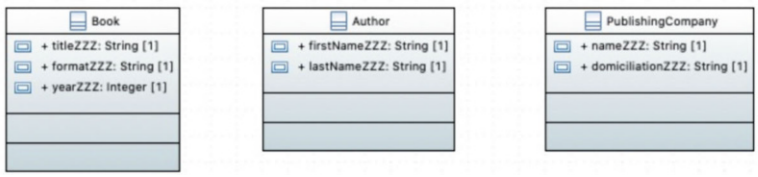

Fig. 5 PapyGame: Hangman example

accumulate points. Figure 6 shows an example. It presents the associated UML diagram containing a set of classes connected with the generalization relationship. The goal of this level is to help players/students associate the right attributes and operations to the right class in the hierarchy. This is done using a drag-n-drop facility. An incorrect user selection (moving an operation into a class that is not the one that should contain it) adds a part of the hangman's body (lower part of Fig. 6). If the players manage to place all operations correctly without the body of the hanged person being completely displayed, they win. The number of gold coins and XP is calculated according to the number of errors (bad drag-n-drops). If the hangman's body is completely displayed, the player loses, and the next level stays unlocked. As a consequence, they will have to play this level again. Whether they won or lost, after the completion of the game, PapyGame players are returned to the Dashboard view.

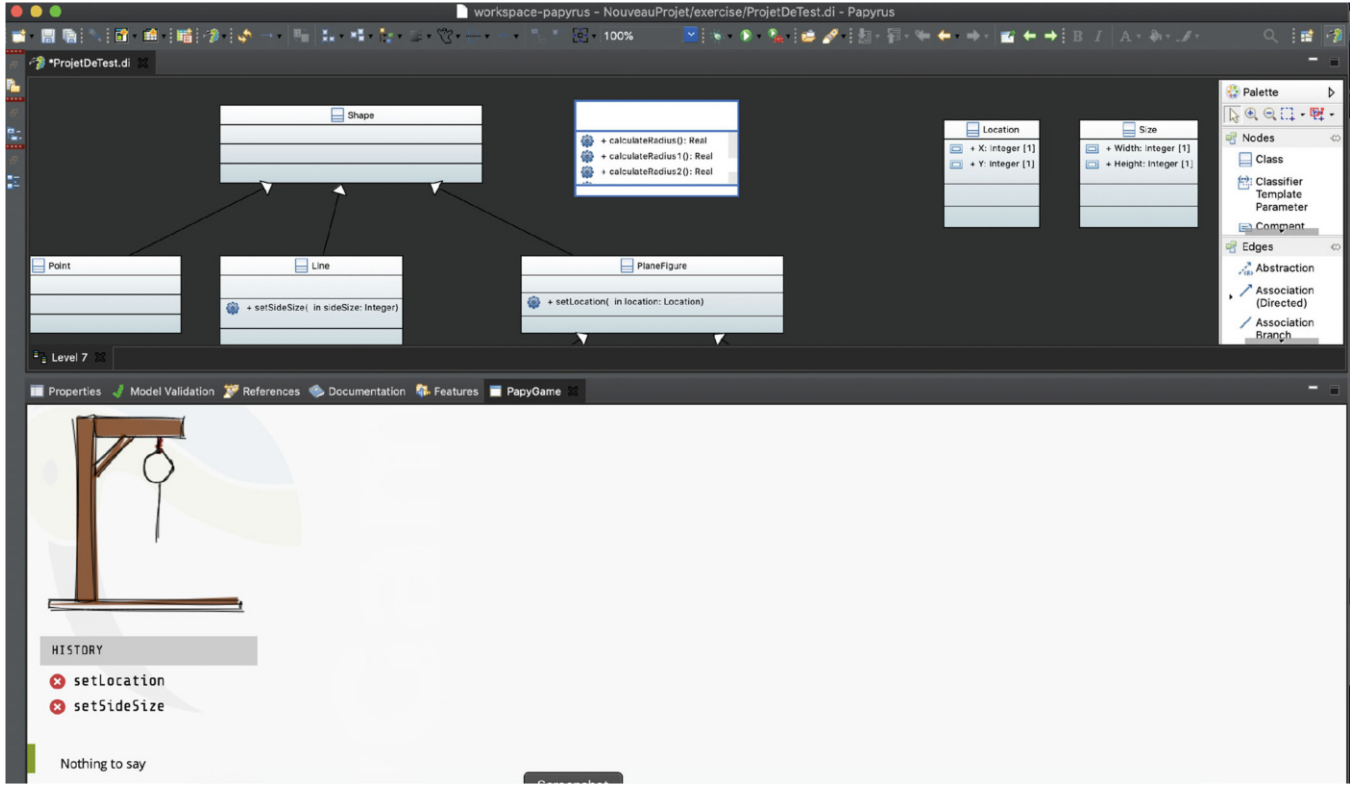

Fig. 6 PapyGame Exercise Example

4 Engineering the Gamification Design Framework (GDF) with MPS

The GDF is composed of a set of languages, each of which defined to cover the artifacts constituting the gamification stack. A graphical representation of this stack is shown in Fig. 7 [6]: it is composed of a set of layers that will be referred to as game modeling layers, namely, GML, GaML, and GiML. They represent incremental refinements/specializations of gamification concepts, from higher to lower levels of abstraction, respectively. The remaining layers, i.e., GsML and GadML, are called *utility layers* and can be defined on top of any of the game modeling ones.

GDF conveys a gamification design process that reflects widely adopted practices in the state of the art and practice of the field [2, 25] (see also Sect. 2). Taking inspiration from this process, GDF provides different modeling languages for specifying the main game components, i.e., game elements, and how they interact to build up a gameful application, that is, mechanics. Such components are progressively refined to reach implementation code for a target gamification engine that copes with game instances execution. For this purpose, we selected a specific gamification engine [26] based on DROOLS rule engine.[6] It is an open-source component and exposes its main functionalities as services (Open APIs) that are used by GDF. Notably, services include supporting the definition and deployment of games, accessing information about the game and player state, and supporting the configuration of notifications for communicating game results to the players.

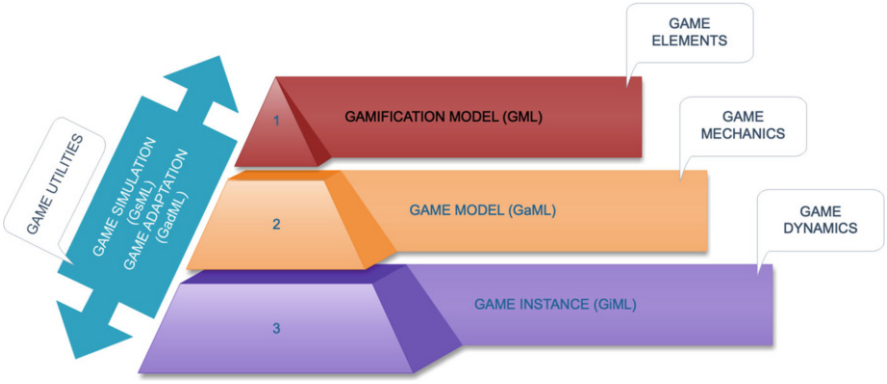

Fig. 7 Languages of the Gamification Design Framework

[6] https://www.drools.org/.

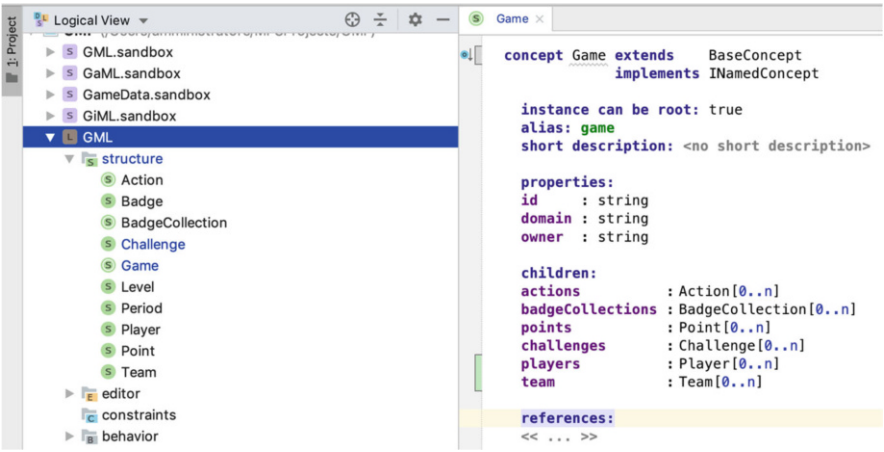

Fig. 8 The Gamification Model Language (GML)

The *Gamification Modeling Language (GML)* is used to represent the set of core elements essential to describe a gameful system (i.e., Point, Badge, Action, etc. introduced in Fig. 2). Figure 8 shows an excerpt of the main GML concepts: a `Game` concept is composed of a set of properties (i.e., `id`, `domain`, and `owner`) that characterize a specific gameful solution, and a set of *children* concepts that allow to specify the main game elements, that is, the fundamental ingredients of a gamified application. GML conforms to the MPS base language and provides the basic gamification building blocks. Other languages (GaML, GiML, etc.) are derived as lower abstraction levels.

In this respect, a *game designer* should extend/refine GML concepts every time there is a need to introduce new game elements or mechanics.

The *Game Model Language (GaML)* extends GML with concepts used to define concrete game descriptions. As shown in Fig. 9, through GaML, the designer can specify how the game components are assembled to create an application into a `GameDefinition`. Notably, the concept of `Point` in GML is specialized in `skillPoint` and `experiencePoint`, to distinguish between points gained by means of specific activity goals and points gained due to the progression through the game, respectively. Moreover, `dataDrivenAction` and `evenDrivenAction` are exploited to recognize activities based on data (i.e., modeling task completed.) or on events (i.e., surveys filled). In a similar manner, the `Challenge` concept coming from GML is refined through, e.g., `PlayerChallenge` and `TeamChallenge`, to distinguish between challenges intended to be completed individually and the ones to be accomplished as groups of players, respectively.

GaML is generic enough to enable the reuse of the defined gamification concepts into multiple development scenarios (e.g., the distinction between the types of actions and points).

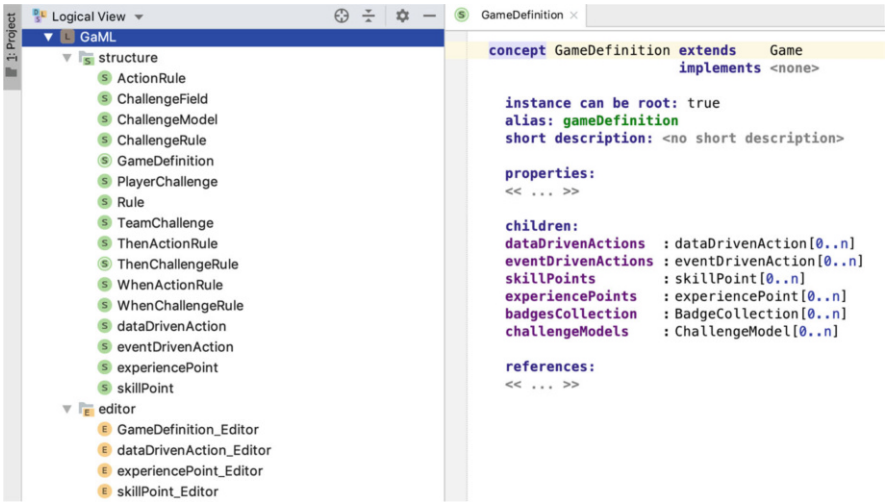

Fig. 9 GameDefinition concept of the GaML

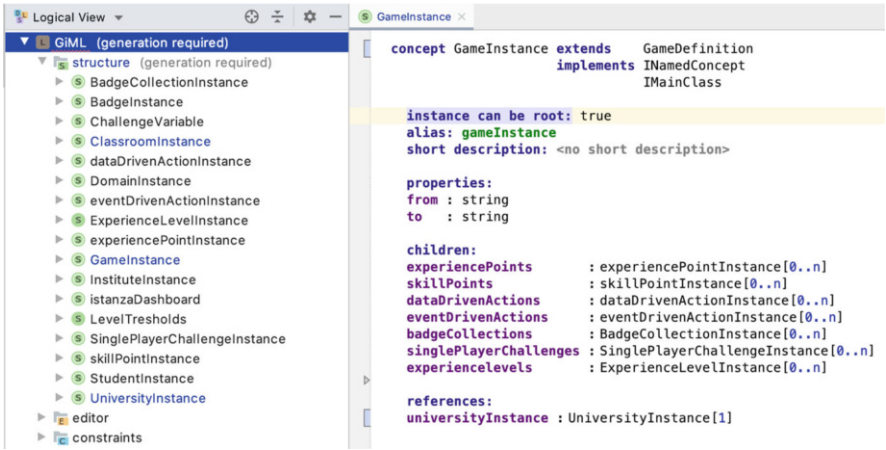

Fig. 10 GameInstance concept of the GiML

A game instance is a `GameDefinition`, as prescribed in GaML, opportunely instantiated to be run by the gamification engine. In general, an instantiation consists of the specification of the players/teams involved in the game; hence, one or more instances of multiple games may run concurrently by means of the same engine. The *Game Instance Model Language (GiML)* binds game definitions coming from GaML with instantiation details, as depicted in Fig. 10. In particular, the `universityInstance` defines *teams* and *players* that play in a certain instance of a game. GiML also supports for single-player *challenges*.

```
concept GameSimulation extends    BaseConcept
                       implements INamedConcept

  instance can be root: false
  alias: <no alias>
  short description: <no short description>

  properties:
  << ... >>

  children:
  listOfExecutions : SingleGameExecution[1..n]

  references:
  game : GameDefinition[1]
```

Fig. 11 GameSimulation concept of the GsML

`singlePlayerChallenges` demand to players the fulfillment of a specific goal, whose attainment requires a prolonged individual commitment, typically within a limited period. In order to confer a feeling of progress and mastery, GDF supports the definition of *Levels*. Levels are always defined in association with a specific *Point Concept*. For this in GiML, we have identified two types of levels: `skillLevels` and `experiencelevels`. The first are related to `skillPoints`, while the second to the `experiencePoints` point concepts. GiML instantiates also the `badgeCollections`. They are used in order to further reward, through a collectible visual representation, the results of a player in terms of specific achievements.

Apart from game modeling languages, GDF provides so-called utility languages. One of them is the Game Simulation Language (GsML), which allows to simulate game scenarios. In particular, a `GameSimulation` is composed of a `GameDefinition` and a set of `SingleGameExecution` elements, as depicted in Fig. 11. In turn, each game execution is made up of a `Team` and/or a `Player` that can execute an `actionInstance` or a `challengeInstance` (see Fig. 12). In this way, the designer can specify specific game situations and check what state changes are triggered. In this respect, it is important to mention that the target gamification engine[7] provides the necessary features to track the gamification rules triggered during the execution together with the corresponding state changes.

Another utility feature provided by GDF is *adaptation*. This feature leverages specific capabilities of the target gamification engine and, in particular, a recommendation system for generating players' tailored challenges based on game historical data and current status, a mechanism to "inject" new game contents on the fly. With this premise, GadML allows to model those scenarios when a new game content (i.e., a new challenge recommended by the engine) has to be assigned to a specific

[7] https://github.com/smartcommunitylab/smartcampus.gamification.

```
concept SingleGameExecution extends    BaseConcept
                            implements <none>

  instance can be root: false
  alias: singleGameExecution
  short description: <no short description>

  properties:
  << ... >>

  children:
  << ... >>

  references:
  team              : Team[1]
  player            : Player[0..1]
  actionInstance    : ActionInstance[0..1]
  challengeInstance : ChallengeInstance[0..1]
```

Fig. 12 Single execution of a game simulation

```
concept newChallenge extends    GameAdaptation
                     implements <none>

  instance can be root: true
  alias: newChallenge
  short description: <no short description>

  properties:
  << ... >>

  children:
  challengeModel : ChallengeModel[1]
  challengeData  : ChallengeData[1]
  challengeDate  : ChallendeDate[1]

  references:
  << ... >>
```

Fig. 13 newChallenge concept of the GadML

player on the fly. In particular, the `GameAdaptation` concept includes `gameId` and `playerId` parameters for a game adaptation, plus a set of children to specify the new challenge to be injected. As Fig. 13 shows, a `newChallenge` refines a simple game adaptation by defining a `ChallengeModel`, `ChallengeData` (i.e., *bonusScore*, *virtualPrize*, etc.), and `ChallengeDate` (i.e., validity period of time for the challenge).

4.1 MPS Projectional Editors

MPS features "projectional" editing, that is, developers are not editing simple text while providing inputs in MPS; on the contrary, their editing is bound to the abstract syntax tree (AST) inferred by the language definition. In other words, the DSL concepts defined through MPS and used as inputs "activate" specific branches of the AST, and consequently, the editing proceeds by following the available alternatives as per language definition. In this way, the code is always represented as an AST, conforming to the language by construction.

Programs in MPS are represented as instances of concepts, called nodes [27]. In this respect, it is possible to define how the different concepts of a language are visualized to the end user, and each projectional editor provides a representation of the AST with which the user interacts. For each set of concepts in GDF, we have defined editors by means of specific projection rules used to define the desired concrete syntax. These editors can be used by the *game designer* to define the gamification elements, mechanics, and dynamics. At the same time, we added extra editors to define specific game simulations and adaptations.

Figure 14 shows the editor definition for the `GameDeclaration` concept. At the top level, it consists of a collection cell [− . . . −] which aligns a sequence of

```
<default> editor for concept GameDeclaration
  node cell layout:
    [-
    game { name }
      [-
      id : { id }
      -]
      point concepts
      [-
        (/ % pointConcept % /)
          /empty cell: <default>
      -]
      actions
      [-
        (/ % actions % /)
          /empty cell: <default>
      -]
      rules
      [-
        (/ % rules % /)
          /empty cell: <default>
      -]
    -]

  inspected cell layout:
    <choose cell model>
```

Fig. 14 Editor definition for the `GameDeclaration` concept

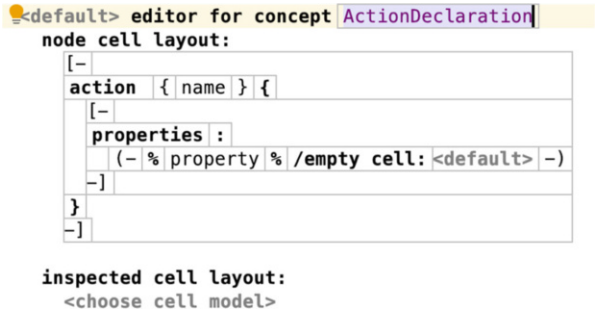

Fig. 15 Editor definition for the `Action` concept

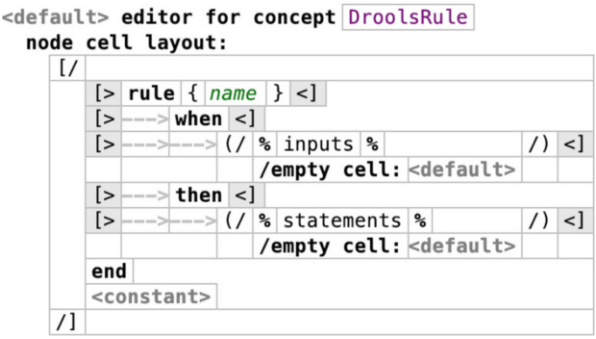

Fig. 16 Editor definition for the `Rule` concept

additional cells. The sequence starts with the game `name` and its related `id`. It continues with the sequence of core elements as `point concepts`, `actions`, and `rules`. All these concepts are the key elements to define the points that each player can receive, the actions that lead to the accumulation of points, and the rules that define the overall game behavior.

Actions (and Points) of a game can be defined at design time by a game designer using specific editors as the one illustrated in Fig. 15.

An action of a game, as the point, is defined by a name and by a set or properties that characterize them. For example, an action can represent the interaction that a player does with the application together with information about the instance when it is executed, while a point represents a counter that is updated with a certain value every time a specific action is executed.

As mentioned before, to execute the game actions done by the player, GDF leverages an open-source Gamification Engine component. This component embeds DROOLS, a state-of-the-art rule engine technology based on reactive computing models [28]. For this reason, GDF expresses the rules of a game using the `DroolsRule` concept that regulates the game behavior. A game designer can use the editor depicted in Fig. 16 to specify `when` a certain rule is triggered and how the different points are accumulated (with the `then` part declaration).

4.2 MPS Generators

The goal of the *generators* in MPS is to go from the business domain (i.e., gamification) to the specific implementation domain (i.e., gamification engine). Each generator is focusing on changing the AST of the specific domain into another AST that is closer to the implementation domain.

In GDF, for each layer introduced in Fig. 7, we have realized a generator as depicted in Fig. 17. This figure represents the internal logic of the GDF as composed of four constituents: (a) Game Model, (b) Game Instance, (c) Game Simulation, and (d) Game Adaptation. Each component is made up of two layers, the former is implemented in MPS, while the latter exploits the gamification engine.[8]

By means of the `Game Model` component, the game designer can specify the core elements of a new game (i.e., points, actions, rules) using the editor depicted in Fig. 14. The GaML generator is used to transform a new game model into the specific elements used by the gamification engine to execute the game. The effect of this transformation is depicted in Fig. 18 where the core elements of a new game are deployed in the gamification engine.

At this point, a gamification designer can instantiate a new game using the `Game Instance` component that exploits the core elements of the game defined in the `Game Model` component and deploys the needed elements in the gamification engine. The `Game Instance` component has been introduced to instantiate the different games starting from the same Game Model. Once the designer have specified the new game, she/he can proceed with the game deployment in the gamification engine. The deployment step is done using the GiML generator able to generate Java code that, when executed, results in the creation of corresponding game instance in the gamification engine.

For example, in the specific case of the teaching modeling domain (described with details in Sect. 3) that we have experimented, the game designer can define the university involved, the set of `SkillPoint` and `ExperiencePoint`, the set of `Actions`, and other information related to the specific game instance (i.e., name, duration, and its description), as depicted in Fig. 19. Starting from a *Game Instance* model, as the one in Fig. 19, and using the generator introduced in Fig. 20, this component is able to deploy the game instance in the target gamification engine calling specific REST APIs[9] supplied by the latter.

GDF also provides support for the simulation of the behavior of a running game and the definition of new game contents and their assignment to a specific player on the fly (i.e., game adaptation). To simulate the defined games, GDF uses the GsML component. A game simulation is defined by a *Player* that should execute

[8]The engine is available in GitHub under the Apache License Version 2.0 https://github.com/smartcommunitylab/smartcampus.gamification and is available as a stand-alone application as well as software as a service (SaaS).

[9]https://dev.smartcommunitylab.it/gamification-v3/swagger-ui.html.

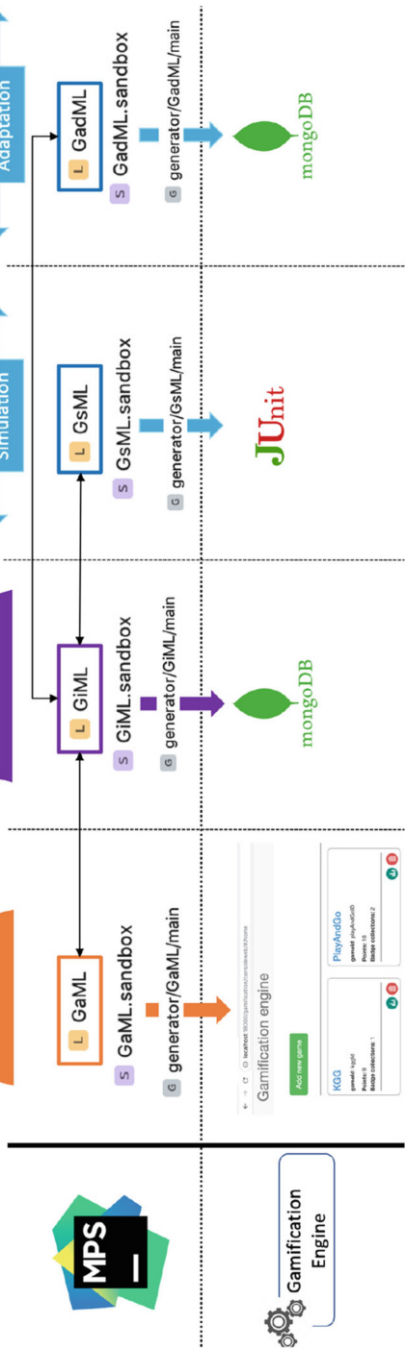

Fig. 17 The GDF internal logic

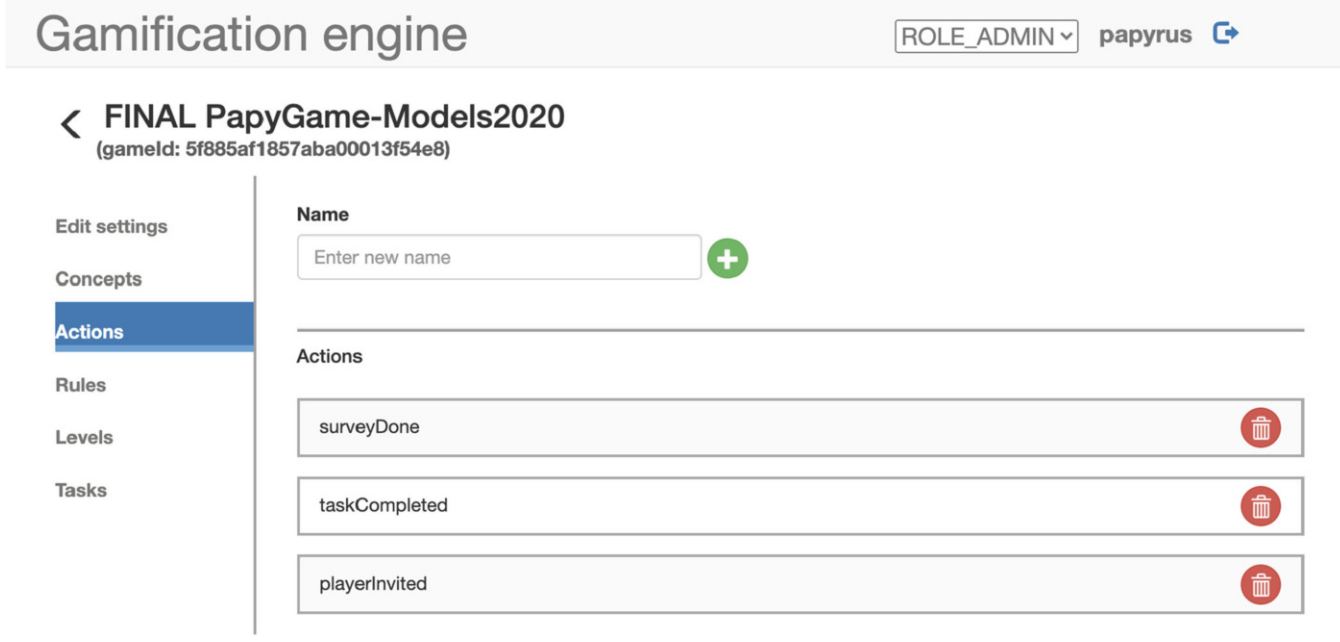

Fig. 18 Game Model in the gamification engine

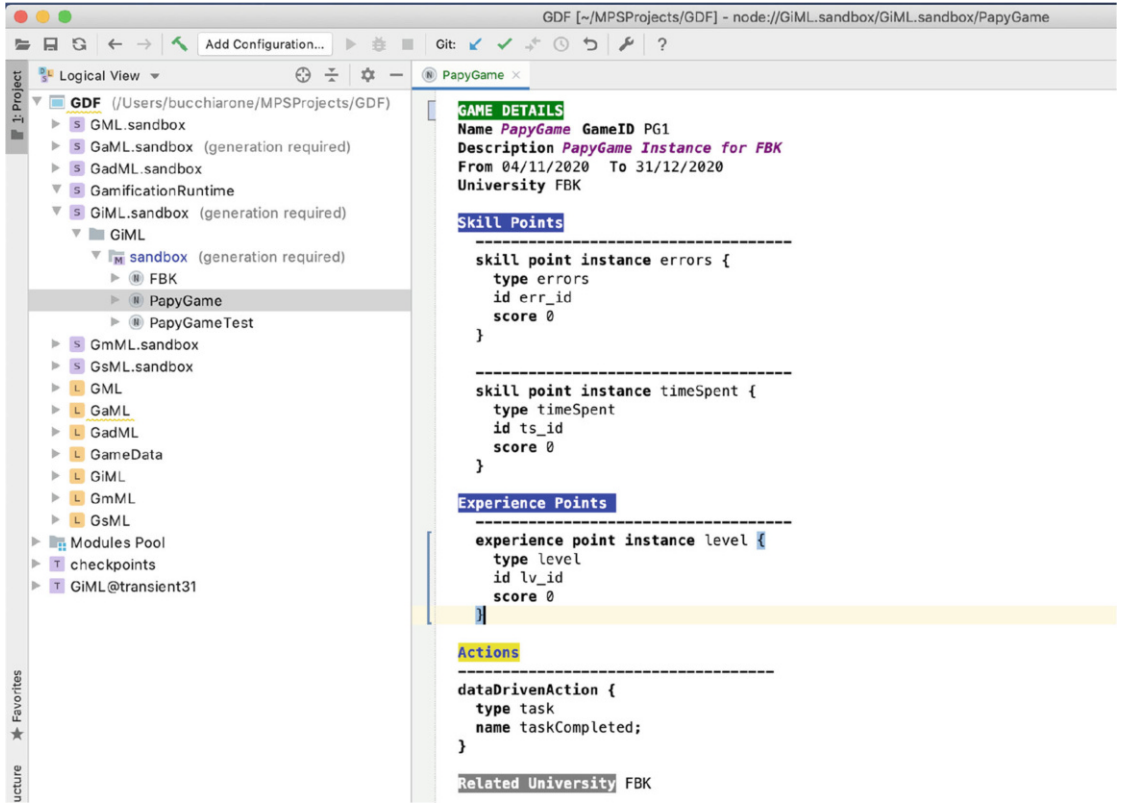

Fig. 19 Game Instance Model for the PapyGame scenario (see Sect. 3)

```
root template
input GameInstance

public class map_gameInstance extends GameTest {

  private PlayerService playerSrv;
  private static final String GAME_ID = "$[gameID]";


  @Override
  public void initEnv() {
    //  add all the Teams defined for this game instance

    $LOOP$ [{
              TeamState team = new TeamState(GAME_ID, "$[teamName]");
              List<string> members = new ArrayList<String>();
              $LOOP$ [members.add("$[playerName]"); ]
              playerSrv.add(team);
            }

  }
  @Override
  public void defineGame() {
    //  add all the GameDefinition elements
    List<String> actions = new ArrayList<String>();

    //  Add Game DataDriven Actions
    $LOOP$ [actions.add("$[actionName]"); ]
    // ......
```

Fig. 20 Game instance generator

```
root template
input GameSimulation

public class map_GameSimulation extends GameTest {
  private static final String GAME_ID = "$[gameID]";
  @Override
  public void defineExecData(List<GameTest.ExecData> execList) {


    $LOOP$ [{
              Map<String, Object> payload = payload = new HashMap<String, Object>();
              $LOOP$ [payload.add("$[dataType]", "$[dataValue]"); ]
              GameTest.ExecData input = new GameTest.ExecData(GAME_ID, "$[actionType]", payload);
              // select a player randomly to assign a specific simulation action
              input.setPlayerId("$[playerId]");
              execList.add(input);
            }
}
```

Fig. 21 Game simulation generator

an *actionInstance* or a *challengeInstance*. Starting from a *GameSimulation* model specified using GsML and containing the elements we mentioned earlier, we used the generator introduced in Fig. 21 to derive the corresponding Java code.

In this case, a `jUnit` test, as the one depicted in Fig. 22, is generated. In the specific case, it represents a game simulation where a specific player (i.e., Alice) executes the `taskCompleted` action with 1 `errors` and 20 `timeSpent`.

Finally, whenever a game adaptation (i.e., new challenge) must be executed for a specific player of a team, GDF provides the `Game Adaptation` component. It

```
@Override
public void defineGame() {
     mongo.getDb().drop();
    List<GameConcept> concepts = new ArrayList<~>();
    concepts.add(new PointConcept( name: "moves"));
    concepts.add(new PointConcept( name: "errors"));
    concepts.add(new PointConcept( name: "points"));
    concepts.add(new PointConcept( name: "gold coins"));
    concepts.add(new PointConcept( name: "timeSpent"));
    List<String> actions = Arrays.asList("taskCompleted", "surveyDone", "playerInvited");
    defineGameHelper(DOMAIN, GAME, actions, concepts);
    try {
        loadClasspathRules(GAME,  classpathFolder: "rules/"+GAME);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
@Override
public void defineExecData(List<ExecData> execList) {
   Map<String, Object> d = new HashMap<~>();

    d.put("errors", 1.0);
    d.put("timeSpent", 20.0);
    ExecData data = new ExecData(GAME,  actionId: "taskCompleted",  playerId: "alice", d);
    execList.add(data);
}
@Override
public void analyzeResult() {
    assertionPoint(GAME,  score: 1.0 + 1.0,  playerId: "alice",  conceptName: "gold coins");
    assertionPoint(GAME,  score: 30.0 + 40.0,  playerId: "alice",  conceptName: "points");

}
```

Fig. 22 Excerpt of the jUnit test for the game simulation

is supported by a generator to generate a Java code that leads to the adaptation of a database (as depicted in Fig. 17d) that the gamification engine uses to update the specific game instance.

5 Lessons Learned and Future Investigations

As mentioned in Sect. 2.3, the language engineering process devoted to the creation of GDF constituted an endeavor due to diverse challenges, summarized as follows and discussed in detail after:

- scalability of modeling techniques, and in particular of diagrammatic forms, in the case of systems of logical constraints;
- multiple levels of concepts definitions and corresponding instantiations;
- usability of available multilevel modeling workbenches.

The first problem can be regarded as general and not due to the specific application domain and the language workbench taken into account. Indeed, using diagrammatic representations for complex systems of logical constraints tends to become quickly intractable: modeling sequences of events together with their preconditions and constraints, possibly also including negative statements, makes the size of the specifications to exponentially grow with the number of handled variables/events. We already gained large experience in modeling smart mobility applications and incurred in the same scalability problem we faced for gamification [29]. As a consequence, after some preliminary attempts and the necessary investigation of the state of the art, we decided to opt for a text-based modeling solution. Admittedly, at this point, we could have chosen a different language workbench, notably XText,[10] for trying to implement GDF. However, the language import feature embedded in MPS caught our attention, and we decided to give it a try, without a thorough comparison with possible text-based alternatives.

The second among the listed challenges can be reduced to a separation-of-concerns need [30]: in order to cope with the growing complexity of gameful applications, it is desired to partition their design and development into smaller, typically simpler, sub-problems. More precisely, sub-problems would be (i) the definition of gaming elements and mechanics in general; (ii) the specification of selected elements and mechanics for a certain game; and (iii) the deployment of game elements and mechanics of a specific gamification engine, including the configuration of players and teams. It is worth noting that in order to make this separation effective, consistency support shall also be provided, such that, e.g., it would not be possible to use a game element when not permitted by the mechanics used in the game itself. In this context, the language import feature provided by MPS reveals its criticality: since MPS is based on projectional editing, it is simply not possible to produce ill-formed models. As a direct consequence, since, e.g., models in (ii) originate from the definitions in (i), a mechanism or element defined in (i) can only be used in (ii) as conforming to the language definition. In the same way, a deployment specification in (iii) can be only done as conforming to the definition of the game provided in (ii).

In the MDE literature, separation-of-concerns and consistency management have been widely investigated, and in fact, there exist solutions like multilevel modeling and megamodeling that would suit for GDF purposes. Nonetheless, while from a theoretical perspective those approaches are valid, the same cannot be said about availability of tools. As a matter of fact, in our experience, MPS has been the first workbench delivering language engineering support with a tractable level of complexity, even for persons not necessarily MDE experts. Here it is important to remind that GDF is intended to be extensible to new game elements, mechanics, etc., and to be used by all gamification stakeholders. Therefore, the usability of the language workbench is of critical importance. As mentioned before, the language extension mechanisms provided by MPS allow to import existing languages and extend them

[10] https://www.eclipse.org/Xtext/.

by operating limited refinements, both for new concepts, new constraints, and so forth. At the same time, consistency management is implicitly built by means of inheritance relationships.

As expected, also MPS requires a training period, and the learning curve might be steep. In our experiences, people with programming backgrounds might have some advantage in the learning phase, while MDE experts might find some of the features as unexpected and/or counterintuitive. Indeed, MPS exploits a precise set of tools that need to be carefully understood in order to be able to use the workbench effectively. In other words, while MPS demonstrates to be fairly simple to use for entry levels, when users require advanced features, the workbench exposes a rather complex and extensive set of details that need to be taken care of. In this respect, the vast availability of tutorials and examples falls short, because a more conceptual description about how different portions of the workbench are kept together would be required. Moreover, the need for a language debugger increases with the size of the project; in fact, in most cases, a language engineer understands that a specification is wrong due to some unexpected behavior of the resulting artefacts, while the reason of the malfunctions remains completely hidden.

For the current version of GDF, we expect gamification stakeholders to optionally define new game elements and mechanics, while the typical use would be to instantiate existing mechanics and elements in corresponding game definitions. In turn, game definitions would be exploited to define gameful application deployments on a selected gamification engine. Moreover, GDF provides languages to define game monitoring and adaptations at runtime. While in most cases it is enough to define game elements and state change rules, the specifications might grow in their complexity and require fine-tuning for which stakeholders would require MPS experts' help. In this respect, we are planning future refinements/extensions of GDF to make the specification of gameful applications simpler. Notably, we would abstract from the MPS workbench itself by means of, e.g., a web interface for game definitions; moreover, we would like to investigate easier ways of specifying detailed game behaviors, especially to relieve stakeholders of the burden due to hand-tuning the generators embedded in GDF.

6 Conclusions

Gamification is increasingly gaining popularity as a tool to promote engagement in target human activities. In this respect, gameful applications development is facing growing complexity that requires adequate design and deployment support. In this chapter, we presented GDF, a framework for the design and deployment of gameful applications. In particular, GDF is made up of domain-specific languages allowing for stepwise refinement of application definitions, from higher levels of abstraction to implementation code to be run on a gamification engine.

GDF has been engineered by using MPS due to three main reasons: the need to provide text-based DSLs, the availability of language extension mechanisms

conveying consistency management between abstraction layers, and the provision of generators to automatically derive implementation code.

GDF has also been validated against multiple case studies in diverse domains, notably education, smart mobility, and training in modelling. In this respect, MPS demonstrated powerful capabilities but also a steep learning curve that could be unacceptable for non-software engineers. In this respect, one of the main future research directions we are pursuing is the integration of simplified user interfaces, e.g., dashboards, to alleviate the complexity of game definitions for GDF users.

References

1. Seaborn, K., Fels, D.I.: Gamification in theory and action: a survey. Int. J. Hum.-Comput. Stud. **74**, 14–31 (2015)
2. Deterding, S., Dixon, D., Khaled, R., Nacke, L.E.: From game design elements to gamefulness: defining "gamification". In: Lugmayr, A., Franssila, H., Safran, C., Hammouda, I. (eds.), *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, MindTrek 2011, Tampere, September 28–30, 2011*, pp. 9–15. ACM, New York (2011)
3. Hanus, M.D., Fox, J.: Assessing the effects of gamification in the classroom: a longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. Comput. Educ. **80**, 152–161 (2015)
4. TechnologyAdvice.com. Compare 120+ gamification platforms. https://technologyadvice.com/gamification/ (2019)
5. Bucchiarone, A., Cicchetti, A., Marconi, A.: GDF: a gamification design framework powered by model-driven engineering. In: Burgueño, L., Pretschner, A., Voss, S., Chaudron, M., Kienzle, J., Völter, M., Gérard, S., Zahedi, M., Bousse, E., Rensink, A., Polack, F., Engels, G., Kappel, G. (eds.) 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, September 15–20, 2019, pp. 753–758. IEEE, New York (2019)
6. Bucchiarone, A., Cicchetti, A., Marconi, A.: Exploiting multi-level modelling for designing and deploying gameful systems. In: Kessentini, M., Yue, T., Pretschner, A., Voss, S., Burgueño, L. (eds.) 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2019, Munich, September 15–20, 2019, pp. 34–44. IEEE, New York (2019)
7. S. Deterding, M. Sicart, L.E. Nacke, K. O'Hara, D. Dixon, Gamification: using game-design elements in non-gaming contexts. In: Tan, D.S., Amershi, S., Begole, B., Kellogg, W.A., Tungare, M. (eds.) Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Extended Abstracts Volume, Vancouver, BC, May 7–12, 2011, pp. 2425–2428. ACM, New York (2011)
8. García, F., Pedreira, O., Piattini, M., Cerdeira-Pena, A., Penabad, M.R.: A framework for gamification in software engineering. J. Syst. Softw. **132**, 21–40 (2017)
9. Lee, J.J., Hammer, J.: Gamification in education: what, how, why bother? Acad. Exchange Quart. **15**(2), 2 (2011)
10. Zichermann, G., Cunningham, C.: Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps, 1st edn. O'Reilly Media, Inc., Sebastopol (2011)
11. Hugos, M.: Enterprise Games: Using Game Mechanics to Build a Better Business. O'Reilly Media, Inc., Sebastopol (2012)
12. Bartel, A., Hagel, G.: Engaging students with a mobile game-based learning system in university education. In: 2014 IEEE Global Engineering Education Conference (EDUCON), pp. 957–960 (2014)

13. Toda, A.M., do Carmo, R.M.C., da Silva, A.P., Bittencourt, I.I., Isotani, S.: An approach for planning and deploying gamification concepts with social networks within educational contexts. Int. J. Inf. Manage. **46**, 294–303 (2019)
14. Hunicke, R., Leblanc, M., Zubek, R.: MDA: a formal approach to game design and game research. In: Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence, pp. 1–5. AAAI Press, Menlo Park (2004)
15. Schell, J.: The Art of Game Design: A Book of Lenses. Morgan Kaufmann Publishers Inc., San Francisco, CA (2008)
16. Rodrigues, L.F., Oliveira, A., Rodrigues, H.: Main gamification concepts: a systematic mapping study. Heliyon **5**(7), e01993 (2019)
17. Elverdam, C., Aarseth, E.: Game classification and game design: Construction through critical analysis. Games Culture **2**(1), 3–22 (2007)
18. Kim, B.: Designing gamification in the right way. Libr. Technol. Rep. **51**, 29–35 (2015)
19. Robson, K., Plangger, K., Kietzmann, J.H., McCarthy, I., Pitt, L.: Is it all a game? understanding the principles of gamification. Bus. Horiz. **58**(4), 411–420 (2015)
20. Camerer, C.F.: Behavioral Game Theory: Experiments in Strategic Interaction. Russell Sage Foundation, New York, NY (2003)
21. Walk, W., Görlich, D., Barrett, M.: Design, Dynamics, Experience (DDE): An Advancement of the MDA Framework for Game Design, pp. 27–45. Springer International Publishing, Cham (2017)
22. Bucchiarone, A., Cicchetti, A., Marconi, A.: Towards engineering future gameful applications. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER '20, pp. 105–108. Association for Computing Machinery, New York, NY (2020)
23. Atkinson, C., Kühne, T.: Model-driven development: a metamodeling foundation. IEEE Softw. **20**(5), 36–41 (2003)
24. Bucchiarone, A., Savary-Leblanc, M., Pallec, X.L., Bruel, J.-M., Cicchetti, A., Cabot, J., Gerard, S., Aslam, H., Marconi, A., Perillo, M.: Papyrus for gamers, let's play modeling. In: Guerra, E., Iovino, L. (eds.) MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18–23 October, 2020, Companion Proceedings, pp. 5:1–5:5. ACM, New York (2020)
25. Salen, K., Zimmerman, E.: Rules of Play: Game Design Fundamentals. MIT Press, Cambridge, MA (2004)
26. Kazhamiakin, R., Marconi, A., Martinelli, A., Pistore, M., Valetto, G.: A gamification framework for the long-term engagement of smart citizens. In: IEEE International Smart Cities Conference, ISC2 2016 (2016), pp. 1–7
27. Voelter, M., Lisson, S.: Supporting diverse notations in mps' projectional editor. In: Combemale, B., DeAntoni, J., France, R.B. (eds.) Proceedings of the 2nd International Workshop on the Globalization of Modeling Languages co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, GEMOC@Models 2014, Valencia, September 28, 2014. CEUR Workshop Proceedings, vol. 1236, pp. 7–16. CEUR-WS.org, 2014
28. Herzig, P., Wolf, B., Brunstein, S., Schill, A.: Efficient persistency management in complex event processing: a hybrid approach for gamification systems. In: Theory, Practice, and Applications of Rules on the Web—7th Int. Symp., RuleML 2013, July 11–13, 2013. Proceedings. LNCS, vol. 8035, pp. 129–143. Springer, New York (2013)
29. Bucchiarone, A., Cicchetti, A.: A model-driven solution to support smart mobility planning. In: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '18, pp. 123–132. Association for Computing Machinery, New York, NY (2018)
30. De Lara, J., Guerra, E., Cuadrado, J.S.: When and how to use multilevel modelling. ACM Trans. Softw. Eng. Methodol. **24**(2), 12:1–12:46 (2014)