**Antonio Bucchiarone**
Fondazione Bruno Kessler
Trento, Italy
Email: bucchiarone@fbk.eu

**Antonio Cicchetti**
IDT Department
Mälardalen University
Vasteras, Sweden
Email: antonio.cicchetti@mdh.se

**Annapaola Marconi**
Fondazione Bruno Kessler
Trento, Italy
Email: marconi@fbk.eu

# GDF: Engineering Gameful Applications with JetBrains MPS

## Description:

We present GDF, a **Gamification Design Framework** for designing gamified applications **through model-driven engineering mechanisms [1,2]**. The framework is based on a set of well-defined modelling layers that start from the definition of the main gamification elements, followed by the specification of 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 as to realize a multi-level modelling approach. This approach is implemented by means of **JetBrains MPS**, a language workbench based on projectional editing, and has been validated through two gameful systems in the education domain.

## Why GDF is needed

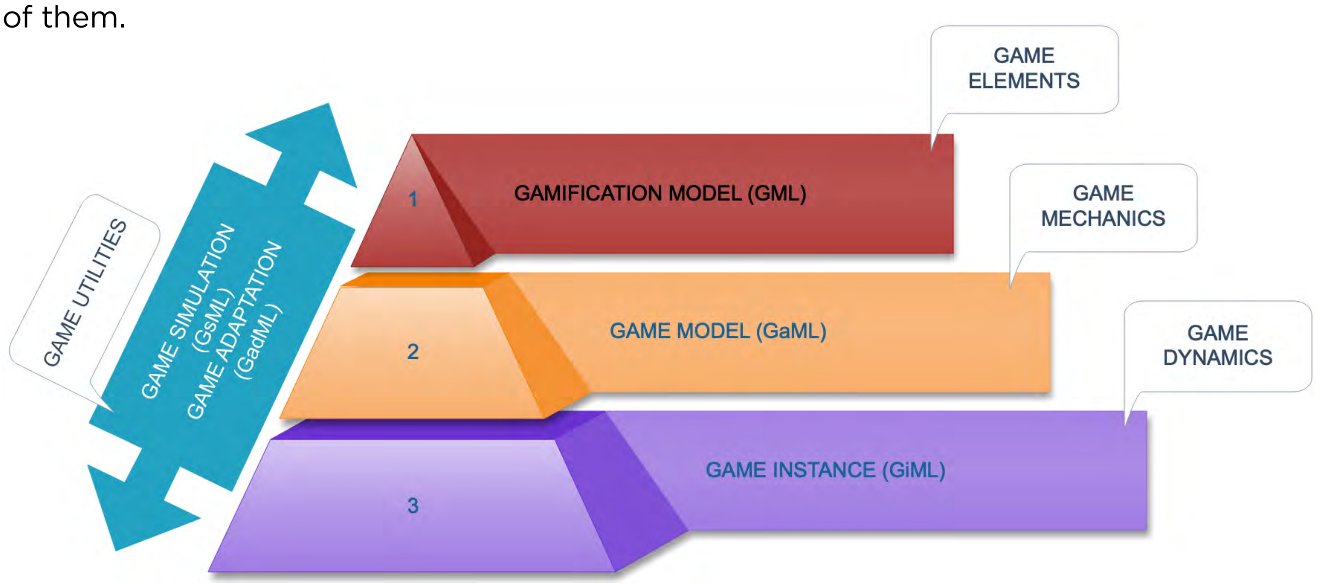
Gamification is gaining popularity in domains that would benefit from an increased engagement of their target users. Gamification applications are found in various contexts, including education, health and environment, and e-banking, just to mention a few. The growing adoption of gamified solutions has made their design and development increasingly complex, owing to, for example, the number and variety of users, and the mission-criticality of some of the applications. Therefore, a rigorous development process is recommended to ensure the gamification project will succeed.[4]

An important challenge in the development and evolution of gameful systems is the *ability to revise or introduce new game elements and mechanics during the system execution.* This is needed in situations when the players' attention decreases or when the impact of the realized system is not as expected. In most of the existing approaches, the design, analysis, and revision of gameful systems requires a lot of development activities often unrelated to each other, with the use of various general-purpose languages, such as Java. The different actors involved, such as a domain expert, a system developer, impact managers, and so on, have been using different languages and tools to execute their tasks with a completely different understanding of the game concepts and their relations. This has led to addressing unexpected game deviations with ad-hoc solutions that are often not reusable, making the monitoring and review of the game mechanics and dynamics a very difficult task.

In order to tackle the complexity of gamification design, we propose the Gamification Design Framework (GDF). **The framework provides a modular approach that can be customized for different gameful systems and is composed of *a well-defined set of languages for designing a game,* its main components, and the behavioural details.** As a first validation, we re-designed and deployed a real gamification application in the education domain.

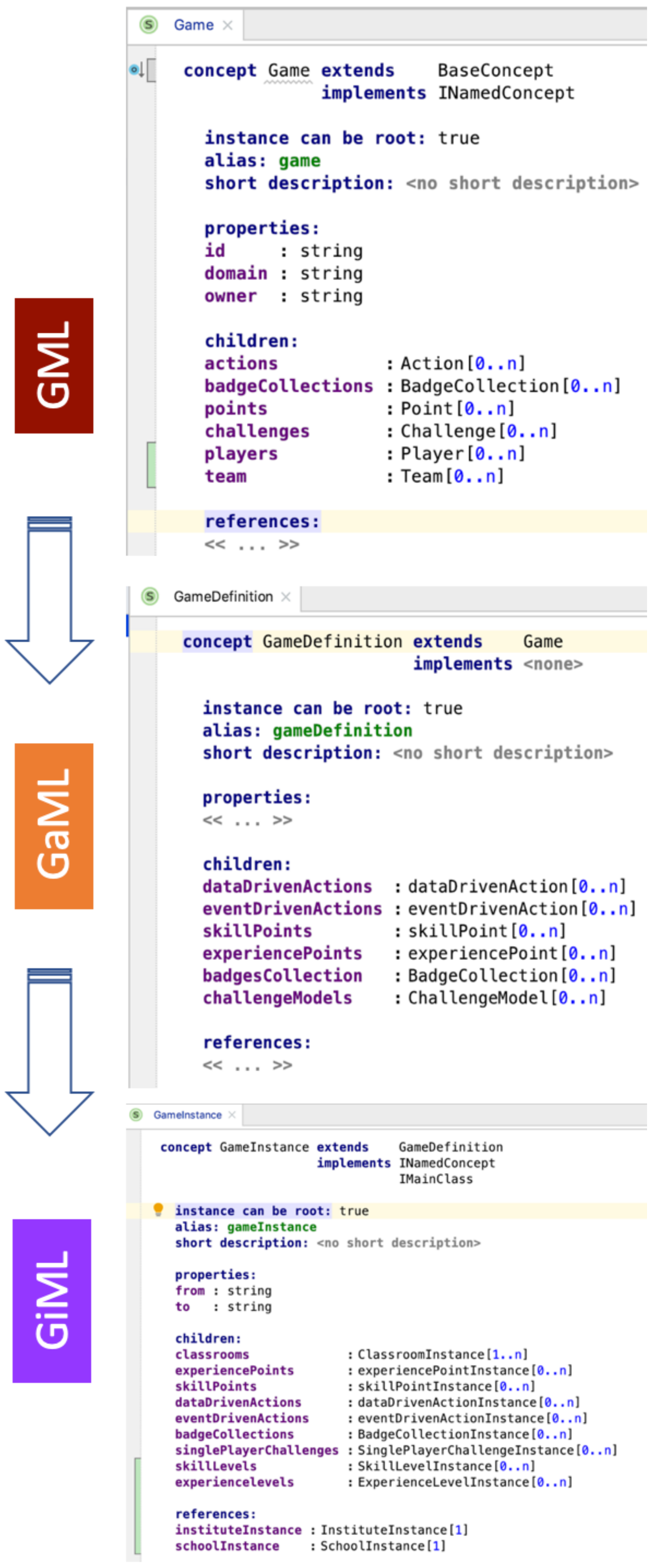
## How GDF works

The GDF architecture has been specified and comprises different modelling layers, each of which is defined on the basis of the layer above. Moreover, utility layers are orthogonal to the others in the sense that they can be defined on the basis of any of them.



The top layer of this architecture is represented by a core language defined to introduce the essential elements to describe a gameful system.

The top component of the architecture shows an excerpt of the concepts that define the abstract syntax of the Gamification Modelling Language **(GML)**. In particular, a Game concept comprises a set of properties (id, domain, and owner) that characterize the specific gameful system, and a set of children representing its core ingredients. Each gamified application must be designed to define a set of game elements devoted to specify how the players should interact with the application. The children part of the Game concept of GML has been defined for this purpose. This core language provides the basic building blocks on top of which all the subsequent modelling layers can be described.



```
concept Game extends      BaseConcept
             implements   INamedConcept

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

  properties:
  id      : string
  domain  : string
  owner   : string

  children:
  actions            : Action[0..n]
  badgeCollections   : BadgeCollection[0..n]
  points             : Point[0..n]
  challenges         : Challenge[0..n]
  players            : Player[0..n]
  team               : Team[0..n]

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

```
concept GameDefinition extends      Game
                       implements   <none>

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

  properties:
  << ... >>

  children:
  dataDrivenActions   : dataDrivenAction[0..n]
  eventDrivenActions  : eventDrivenAction[0..n]
  skillPoints         : skillPoint[0..n]
  experiencePoints    : experiencePoint[0..n]
  badgesCollection    : BadgeCollection[0..n]
  challengeModels     : ChallengeModel[0..n]

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

```
concept GameInstance extends      GameDefinition
                     implements   INamedConcept
                                  IMainClass

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

  properties:
  from : string
  to   : string

  children:
  classrooms                : ClassroomInstance[1..n]
  experiencePoints          : experiencePointInstance[0..n]
  skillPoints               : skillPointInstance[0..n]
  dataDrivenActions         : dataDrivenActionInstance[0..n]
  eventDrivenActions        : eventDrivenActionInstance[0..n]
  badgeCollections          : BadgeCollectionInstance[0..n]
  singlePlayerChallenges    : SinglePlayerChallengeInstance[0..n]
  skillLevels               : SkillLevelInstance[0..n]
  experiencelevels          : ExperienceLevelInstance[0..n]

  references:
  instituteInstance : InstituteInstance[1]
  schoolInstance    : SchoolInstance[1]
```

Based on the core ingredients in GML, the Game Model Language **(GaML)** relies on Game Mechanics and allows the game designer *to design a certain concrete game*. At this level of abstraction the designer can specify in more detail how the game components are assembled to create a gamification application. In particular, *dataDrivenAction* operators act on data (kms, legs, etc.), while *eventDrivenAction* is related to specific events (surveys, check-in, etc.). skillPoint points are used to denote a user's ability in a specific area, while experiencePoint points are used to quantify a player's progression through the game. Finally, *badgeCollection* and *challenge* represent game elements that each player can collect and achieve, respectively. In this way GaML allows the designer to specify game components that are reusable in different gamification scenarios. Notably, the abstract concept *experiencePoint* can take the concrete forms of *pedibus_distance* or *Walk_Km*, depending on the target application.

A gamification engine is responsible for the execution of one or more instances of multiple games that may run concurrently. Therefore, we introduced GiML, a language that relies on the *Game Dynamics* and is used to specify the instantiation of the different games originating from the same *GameDefinition* as defined in GaML. 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. In particular, the classrooms define teams and players that play in a certain instance of a game in the education domain.

When a game instance is running, the *game state* changes whenever one of the mechanics defined in the game (such as score update, challenge achievement, etc.) is used. In particular, this means that one of the game rules defined using GaML is executed in a specific instance defined through GiML. Therefore, the game state evolves as the right-hand side of the game rules prescribe to manipulate the object base through the gamification engine services. Based on this, our approach provides support for *simulating the behavior of* a *running game* under certain conditions, by means of the **GsML** component. It is part of the **Game Utility** component of the GDF and its core concept is represented by the GameSimulation element.

This concept is composed by a *GameDefinition* and a set of *SingleGameExecution*. In this way, GsML allows modeling specific game scenarios together with triggered mechanisms, and hence simulating specific game state changes (such as for testing purposes).

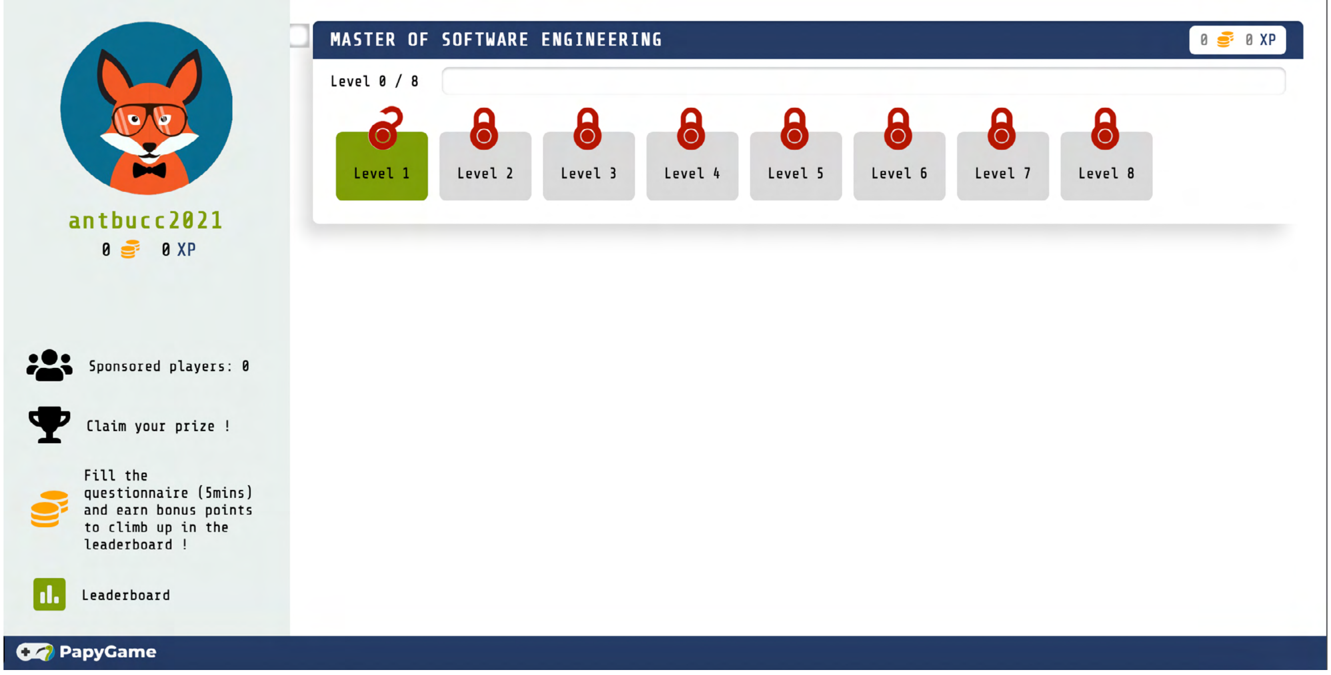
The gamification engine exploited by the GDF includes a Recommendation System (RS) able to generate challenges tailored to each player's history, preferences, and skills. In this context, the **GadML** language is used whenever new game content (such as a new challenge generated by the RS) has to be assigned to a specific player on the fly.

## PapyGame Design with GDF

We have used and validated GDF in diverse application domains, including smart mobility and education. Our latest experience involved using GDF as the core component of a specific gamified software modeling environment called PapyGame. In **PapyGame** the **Papyrus modeling tool** has been extended with gamification features. The objective of the gamification is to help master degree students in Computer Science to *learn specific modeling aspects using Papyrus for UML*.

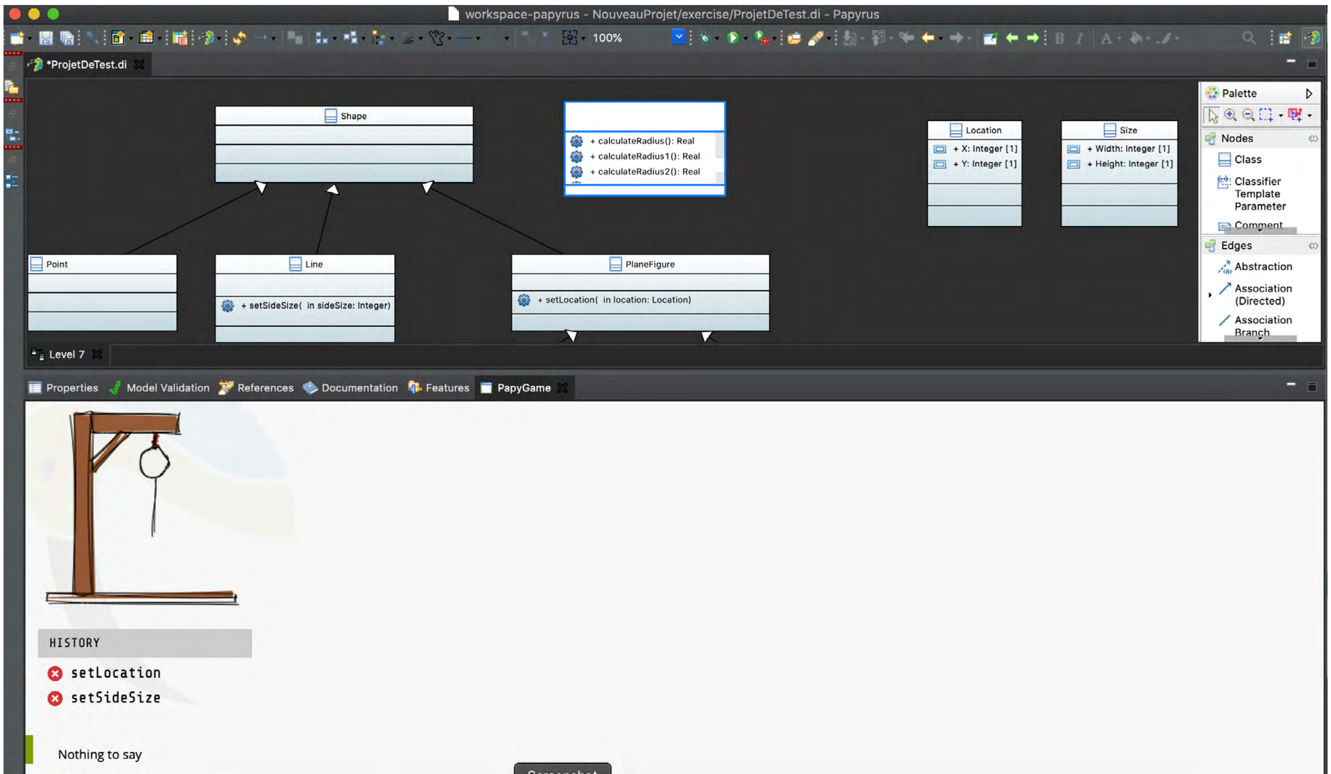
Each *student assignment* in PapyGame is composed of a set of Levels (grouped in Series) that each student should deal with. For each level, an exercise is assigned, and each passed level unlocks the next exercise of the next level. To start a PapyGame session, the player must first enter their login and password. Once connected, PapyGame displays a Dashboard representing the player's series. Each successfully completed level is displayed in green with the corresponding number of gold coins (GC) and 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 exercise has an associated set of point concepts (experience points, gold coins, etc.) and rules. All these aspects are defined **by the teacher by 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.



**GDF is used to automatically save this design task for the teacher and deploy it 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 by exploiting the editors provided by GDF and its related generators.**

Once the *students' assignments* are designed and deployed, the students can select the respective levels and start playing and accumulating points. See the following figure as 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 correct attributes and operations with the correct class in the hierarchy. This is done using a drag-and-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 the example). If the player manages 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 (incorrect drag-and-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 the same level again. Whether the player wins or loses, after the completion of a PapyGame game they are returned to the Dashboard view.



## Conclusions

We presented GDF, a framework for designing and deploying gameful applications. GDF consists of domain-specific languages allowing for stepwise refinement of application definitions, from higher levels of abstraction towards implementation code to be run on a gamification engine.

**MPS was chosen for engineering GDF for 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 modeling. MPS has 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, such as dashboards, to alleviate the complexity of game definitions for GDF users.

## References

[1] Antonio Bucchiarone, Antonio Cicchetti, and Annapaola Marconi. GDF: *A gamification design framework powered by model-driven engineering*. 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, Germany, September 15–20, 2019, pages 753–758. IEEE, 2019.

[2] Antonio Bucchiarone, Antonio Cicchetti, and Annapaola Marconi. *Exploiting multi-level modelling for designing and deploying gameful systems*. 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2019, Munich, Germany, September 15–20, 2019, pages 34–44. IEEE, 2019.

[3] Antonio Bucchiarone, Maxime Savary-Leblanc, Xavier Le Pallec, Jean-Michel Bruel, Antonio Cicchetti, Jordi Cabot, Sebastien Gerard, Hamna Aslam, Annapaola Marconi, Mirko Perillo: *Papyrus for gamers, let's play modeling*. MODELS Companion 2020: 5:1–5:5

[4] Antonio Bucchiarone, Antonio Cicchetti, Annapaola Marconi: *Towards engineering future gameful applications*. ICSE (NIER) 2020: 105–108