GDF: a Gamification Design Framework powered by Model-Driven Engineering

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

Antonio Cicchetti
IDT Department
Mälardalen University
Västerås, Sweden
antonio.cicchetti@mdh.se

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

***Abstract*—Gamification refers to the exploitation of gaming mechanisms for *serious* purposes, like promoting behavioural changes, soliciting participation and engagement in activities, and so forth. In this demo paper we present the Gamification Design Framework (GDF), a tool for designing gamified applications through model-driven engineering mechanisms. In particular, 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 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 to realize a multi-level modelling approach. The 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 and Mobility domains.**

A prototype implementation of GDF and related artefacts are available at the demo GitHub repository: https://github.com/antbucc/GDF.git, while an illustrative demo of the framework features and their exploitation for the case studies are shown in the following video:https://youtu.be/wxCe6CTeHXk.

***Keywords*-Gamification Design Framework, Multi-Level Modelling, Model-Driven Engineering, JetBrains MPS**

I. Introduction and Motivations

Exploiting gaming mechanisms for achieving more serious goals is typically referred to as gamification. Indeed, gamification is more and more establishing itself as a tool for promoting behavioural changes that would be harder to obtain through, e.g., direct prescriptions and rules. Notably, gameful systems find their application into disparate contexts like health, education, environment, and so forth [1]. Gamification relies on a set of consolidated construction elements, that are one or more challenges the achievement of which is rewarded by appropriate prizes distributed to the successful players. The gained awards allow players to advance their status and contribute to their general ranking in the game. These mechanisms are supposed to stimulate players in performing the activities meant to be promoted by the game, and to keep them engaged for long periods of time. As a matter of fact, there exist ready-made gamification solutions able to automatically generate a gameful system based on a minimal set of configuration parameters [2].

A critical aspect for a game to be effective is to keep engaged its players: a too easy/difficult challenge might quickly lose interest and hence make the game goals to fail [3]. In this respect, on the one hand pre-conceived gamification solutions may fall short of malleability needs, especially when dealing with a lot of users and the game combining multiple, interacting challenges. On the other hand, hand-coding the gaming behaviour in a programming language poses other issues, like the abstraction gap between domain experts and the concrete application implementation, and the growing complexity of the programming tasks, especially when considering maintenance and evolution needs. As a consequence, gameful systems development requires well-founded software engineering approaches [4], [5].

Model-Driven Engineering (MDE) [6] is a software engineering methodology that proposes to shift the focus of the development from coding to modelling. The goal is to reduce the complexity of software development by raising the level of abstraction, analyzing application properties earlier, and introducing automation in the process. In fact, models are expected to allow domain experts to reason about a certain solution by means of concerns closer to their area of expertise than to implementation details. Moreover, automated mechanisms, i.e. model transformations, can manipulate those models to evaluate attributes of the application and/or to generate implementation code.

This demo presents the Gamification Design Framework (GDF) [7], a tool for the design and implementation of gameful systems. GDF has been conceived by pursuing three main properties:

- abstraction, the framework shall allow the design of a game in domain-specific terms, that is gamification elements in a certain application domain scenario, rather than focusing on implementation details;
- extensibility, given the characteristics of target applications, the framework shall not constrain the designers in using pre-conceived gamification mechanisms;
- effectiveness, by adopting GDF the designer shall not incur in excessive accidental complexity, comparable to developing the same applications by writing source code.

By going into more details, abstraction is required to allow domain experts to gain more control about the exploited mechanisms and their effects over the target players. In fact, in current state-of-the-art approaches the core business logic has to be typically encoded manually in a rule-/event-based programming language. This makes it difficult for domain-experts (not necessarily programmers) to backtrack monitored game dynamics to corresponding game elements. Additionally, programmers might translate game requirements into erroneous rules causing deviations difficult to detect and understand.

Gamification is intrinsically a creative domain of software applications. Therefore, any development support shall not constrain the solution into pre-conceived boundaries, that is it shall be extensible. In this respect, ready-made solutions could be useful to perform exploratory experiments to understand what mechanisms to use, but would fall short of adaptation possibilities in the long run.

With effectiveness we refer to the concrete gain in adopting the MDE based framework when compared to writing directly implementation code for the gamified application. This is a quite general concern for software engineering in general and MDE in particular, since there exists always the risk of over-engineering the system under study, especially when automation, re-use, and maintenance aspects are irrelevant. Gamification could potentially benefit e.g. of automated code generation and re-use, but these should not come at the price of overly complex system designs.

By means of the Gamification Design Framework we propose a structured approach and a well-defined set of languages for designing a game, its main components, and the behavioural details. GDF is structured in the sense that it provides several abstraction layers, defining: on top the general game elements and mechanics; based on these, the next layer specifies a gamification model, i.e. a specific way of combining the elements and mechanics to build-up a game; starting from the gamification model, in the next layer GDF allows to declare concrete instances of the game elements and hence enables to deploy and run a game by means of a gamification engine. Each layer is defined by means of a domain-specific language where a lower level of abstraction layer instantiates and possibly refines entities pertaining to the layer(s) above. In this respect, GDF realizes a multi-level modelling approach [8], [9], [10]. The multi-level modelling approach implicitly ensures type correctness, since the instantiation of language elements has to be done such that to produce well-formed models.

The GDF implementation of the gamification design and deployment is realised by means of JetBrains MPS (briefly, MPS)[1]. MPS is a meta-programming framework that can be exploited as modelling languages workbench, and provides projectional editors. Moreover, it smoothly supports

[1]http://www.jetbrains.com/mps/

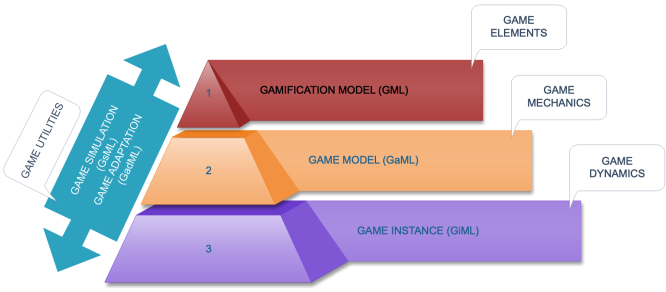

Figure 1. The Gamification Design Framework (GDF)

languages embedding, such that the proposed multi-level definition of gamification solutions is easily implemented. Additionally, since one of the main characteristics of game definitions is being collections of rules and constraints, GDF languages adopt text-based concrete syntaxes. In fact, graphical languages tend to not scale with the complexity of the rules. Eventually, GDF leverages MPS generation features to automatically derive Java implementation code to be run through a specific gamification engine. To demonstrate the features of GDF, we re-design and deploy a real gamification application in the education domain.

The remainder of the paper is organized as follows: the next Section presents the Gamification Design Framework (GDF) with all its languages, while Section III shows the concrete gamification application used to demonstrate the framework and its prototype implementation.

II. The Gamification Design Framework prototype

The GDF adopts a modular architecture composed by several modelling layers, each of which approaching gamification at a different level of abstraction. A graphical representation of this stack is shown in Figure 1 [7]. It is worth noting that some of the layers that we will call *game modelling layers*, namely GML, GaML, and GiML in the figure, represent incremental refinements/specializations of gamification concepts, from higher to lower levels of abstraction, respectively. Instead the remaining layers, i.e. GsML and GadML, are so called *utility layers* and can be defined on top of any of the game modelling ones (their purpose will be better clarified later in this section).

The modular approach provided by GDF implicitly conveys a gamification design process that reflects widely adopted practices in the state-of-the-art and practice of this field [11], [12]. In particular, it provides different modelling 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.

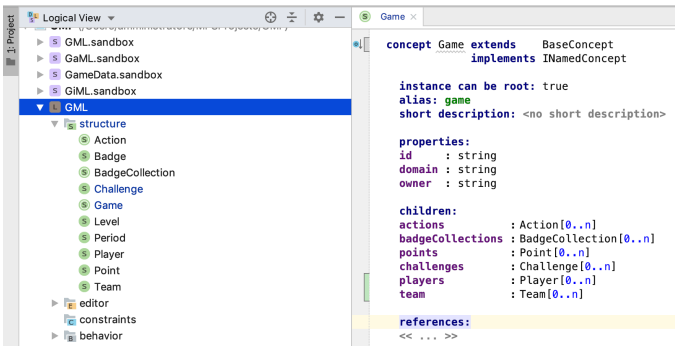

Figure 2. The Gamification Model Language (GML).

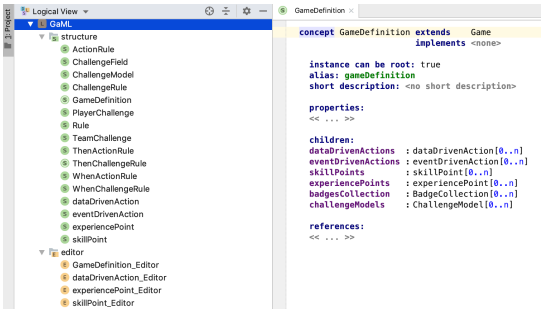

Figure 3. GameDefinition Concept of the GaML.

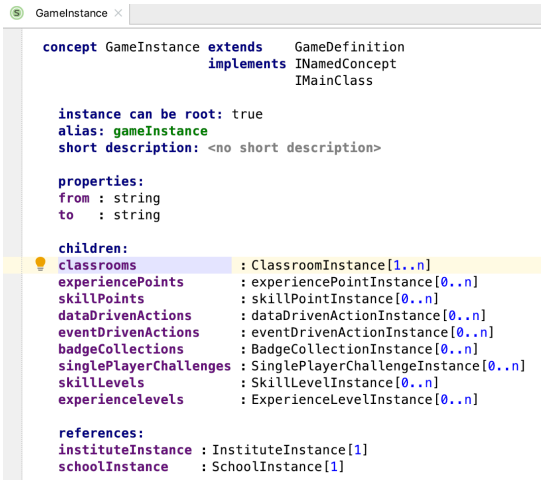

Figure 4. GameInstance Concept of the GiML.

A. The Gamification Model Language (GML)

The top layer of GDF represents a set of core elements essential to describe a gameful system and is called Gamification Modelling Language (GML). Figure 2 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 characterise 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 building blocks on top of which all the subsequent modelling layers can be described. In this respect, a game designer should extend/refine GML concepts every time there is a need of introducing new game elements or mechanics.

B. The Game Model Language (GaML)

Starting from the primitive gamification elements it is possible to define a specific concrete game. For this purpose, the Game Model Language (GaML) instantiates the concepts defined in GML as a concrete game description. As depicted in Figure 3, the designer can specify further 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., kms, legs, etc.) or on events (i.e., surveys, check-in, etc.). 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). Nonetheless, it allows to automatically generate part of the implementation for the application on a specific target gamification engine, notably the set of challenges, the actions, the points, and so forth[2]. More specifically, the outputs of GaML code generation are Java classes ready to be deployed in the gamification engine.

C. The Game Instance Model Language (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 Figure 4. In particular, the `classrooms` defines *teams* and *players* that play in a certain instance of a game.

Analogously to GaML, GiML includes code generation features that generate Java classes corresponding to the game instances defined through the `GameInstance` concept.

[2]For the sake of space, the details of code generators are not shown in the paper. The interested reader is referred to the GDF GitHub repository

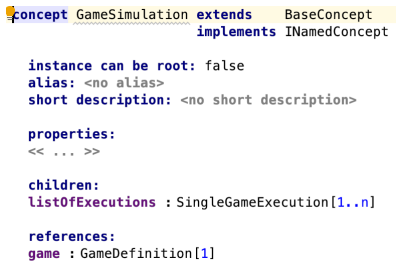

Figure 5. GameSimulation Concept of the GsML.

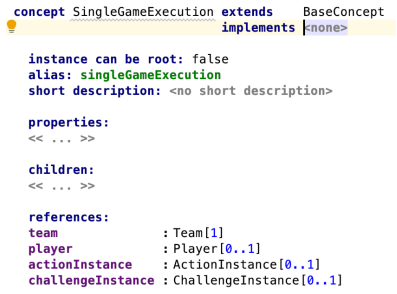

Figure 6. Single Execution of a Game Simulation.

D. The Game Simulation Language (GsML)

Apart from game modelling languages, GDF provides so called utility languages. One of them is the Game Simulation Language (GsML), that allows to simulate game scenarios. In particular, a `GameSimulation` is composed of a `GameDefinition` and a set of `SingleGameExecution` elements, as depicted in Figure 5. In turn, each game execution is made-up of a `Team` and/or a `Player` that can execute an `actionInstance` or a `challengeInstance` (see Figure 6). 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 [3] provides the necessary features to track the gamification rules triggered during the execution together with the corresponding state changes.

E. The Game Adaptation Language (GadML)

Another utility feature provided by GDF is adaptation. This feature leverages specific capabilities of the target gamification engine: 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, the GadML language allows to model those scenarios when a new game content (i.e., a new challenge recommended by engine) has to assigned to a specific player on-the-fly. In particular, the `GameAdaptation` concept includes `gameId` and `playerId` parameters for a game adaptation, plus

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

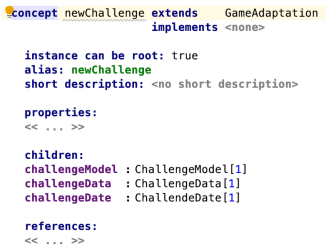

Figure 7. newChallenge Concept of the GadML.

a set of children to specify the new challenge to be injected. As Figure 7 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).

GadML is equipped with an apposite code generator that allows to inject a corresponding game adaptation as per the `GameAdaptation` definition.

III. Demonstrator Details

GDF is demonstrated by means of a real gamified application called *Kids-go-Green* (KGG) [13][4]. KGG is an application for supporting active and sustainable mobility habits, both at personal and collective level, and targeting younger citizens (elementary schools students). KGG is based on a virtual journey that can be travelled by players using the sustainable kilometres collected in real life. Moreover, the virtual journey can be customized according to the interests and the educational needs of the teachers for specific classrooms. Every day each class accesses the KGG Web application and fills the *Mobility Journal*. The children indicate the way they reached school that day by expressing a specific *sustainable transportation mode* (i.e., by foot, bike, walking bus, or school bus). The number of kilometres travelled each day by the children in their trips to school contributes to the progress on a *virtual journey*. KGG also includes class-level and school-level *challenges*: upon completion of a specific objective (e.g., *zero emission day*, *no-car week*, etc.).

Using GDF, a gamification designer can design a new game defining the set of teams (i.e., schools and classrooms) and players (i.e., students) participating in the virtual journey. The specific `Institute` with its respective `Schools` and `Classrooms` are specified with all the needed information (see Figure 8 label 1). Starting from these models, all the game elements (i.e., points, actions, badges, challenges, etc..) that regulate the game behavior can be defined using a specific editor (see Figure 8 label 2). It has been created using all the the fields of the `GamificationInstance` concept of the GiML language (see Figure 8 label 3).

[4]https://kidsgogreen.eu

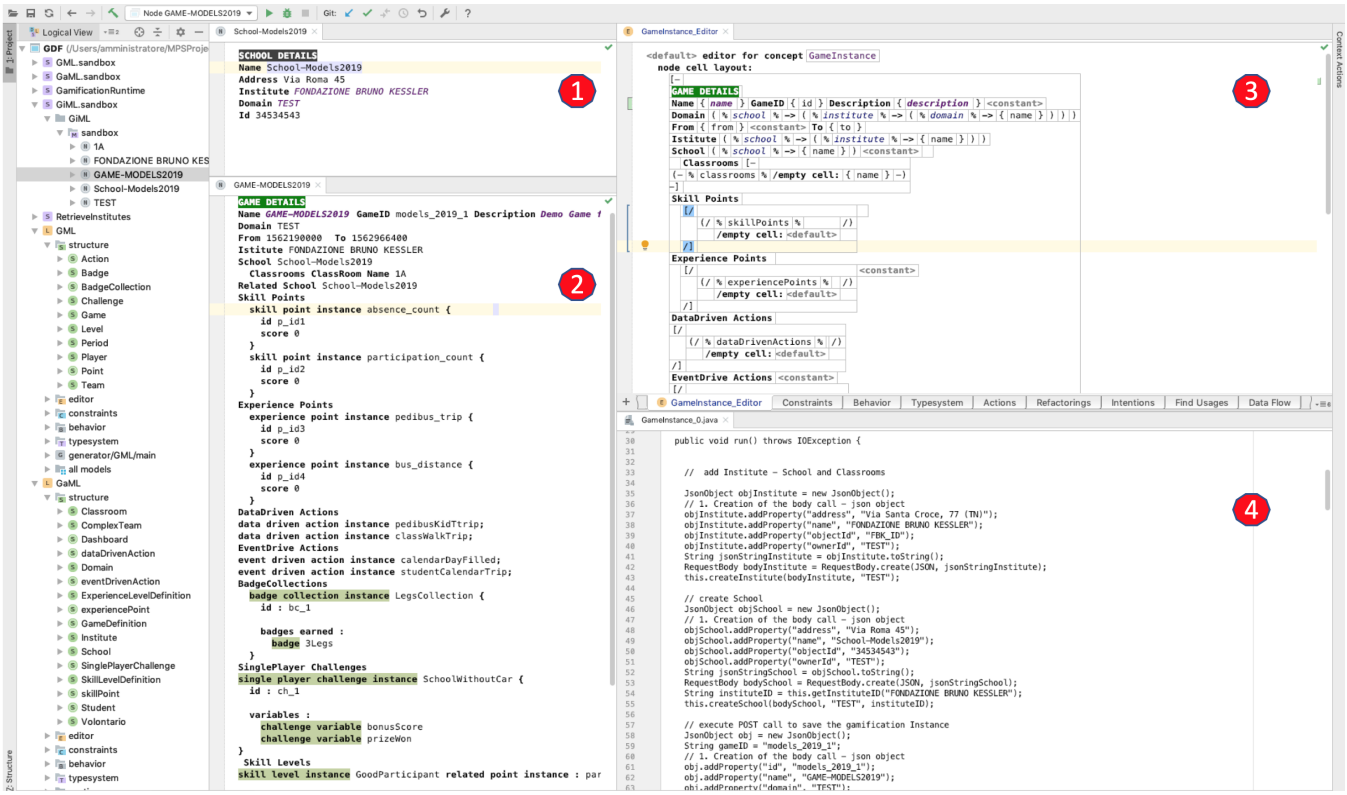

Figure 8. Game Definition Steps using GDF.

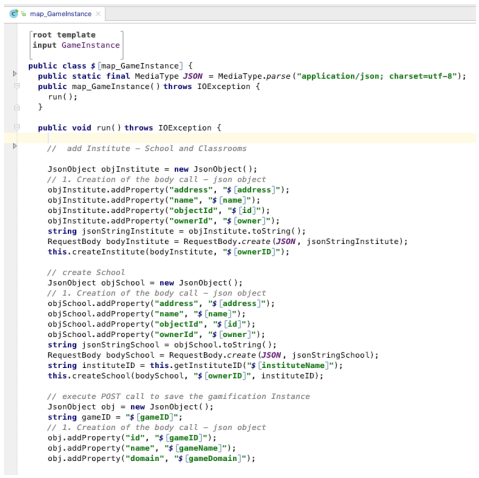

Figure 9. GameInstance generator for a new game.

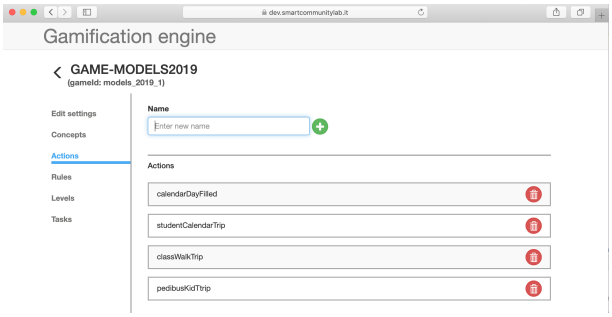

Figure 10. Deployed Game Instance in the Gamification Engine.

Exploiting the code completition facility provided by MPS, the generated editor helps the designer to define all the game elements within the visibility scope of the overall game design.

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 depicted in Figure 9. When executed, the generated Java code results in the creation of corresponding game instance in the gamification engine (see Figure 10). GDF also provides support for the simulation of the behaviour of a running game and the definition of new game contents and their assignment to a specific player on-the-fly.

To see all these aspects in action, we have realized a video, available at the following url:https://youtu.be/wxCe6CTeHXk. It shows: (1) the specification and the deployment of a new game; (2) the specification of a game simulation that allows to simulate game scenarios before the real deployment and, (3) the specification of new game contents and its assignment to a specific player on-the-fly.

ACKNOWLEDGMENT

This work was partially funded by the EIT Climate KIC projects CLIMB Ferrara and SMASH.

REFERENCES

[1] Jonna Koivisto and Juho Hamari. The rise of motivational information systems: A review of gamification research. *International Journal of Information Management*, 45:191 – 210, 2019.

[2] Compare 120+ gamification platforms. https://technologyadvice.com/gamification/. Accessed: April 2019.

[3] Mihaly Csikszentmihalyi and Isabella Selega Csikszentmihalyi. *Optimal experience: Psychological studies of flow in consciousness*. Cambridge university press, 1992.

[4] Oscar Pedreira, Félix García, Nieves Brisaboa, and Mario Piattini. Gamification in software engineering - a systematic mapping. *Information and Software Technology*, 57:157 – 168, 2015.

[5] Benedikt Morschheuser, Lobna Hassan, Karl Werder, and Juho Hamari. How to design gamification? a method for engineering gamified software. *Information and Software Technology*, 95:219 – 237, 2018.

[6] Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, February 2006.

[7] Antonio Bucchiarone, Antonio Cicchetti, and Annapaola Marconi. Exploiting multi-level modelling for designing and deploying gameful systems. In *In IEEE/ACM 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), September 1520, 2018, Munich, Germany*, 2019. To appear.

[8] Colin Atkinson and Thomas Kühne. Model-driven development: A metamodeling foundation. *IEEE Softw.*, 20(5):36–41, September 2003.

[9] Juan de Lara, Esther Guerra, and Jesús Sánchez Cuadrado. Model-driven engineering with domain-specific meta-modelling languages. *Software and System Modeling*, 14(1):429–459, 2015.

[10] Juan de Lara, Esther Guerra, and Jesús Sánchez Cuadrado. When and how to use multilevel modelling. *ACM Trans. Softw. Eng. Methodol.*, 24(2):12:1–12:46, 2014.

[11] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart E. Nacke. From game design elements to gamefulness: defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, MindTrek 2011*, pages 9–15, 2011.

[12] Katie Salen and Eric Zimmerman. *Rules of play: game design fundamentals*. MIT Press, 2004.

[13] Annapaola Marconi, Gianluca Schiavo, Massimo Zancanaro, Giuseppe Valetto, and Marco Pistore. Exploring the world through small green steps: improving sustainable school transportation with a game-based learning interface. In *Proceedings of the 2018 International Conference on Advanced Visual Interfaces, AVI 2018*, pages 24:1–24:9, 2018.