MODEL DRIVEN DEVELOPMENT OF GAMIFIED APPLICATIONS

Piero Fraternali, Sergio Luis Herrera Gonzalez

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milan, 20133, Italy
piero.fraternali@polimi.it, sergioluis.herrera@polimi.it

Abstract

Gamification is defined as the injection of game elements in applications with non-gaming purposes. This technique has shown outstanding results in promoting the engagement and activity on communities of users, in both business and non-for-profits fields. Often, gamification features are added late in the application life-cycle and must be weaved into the existing functions. In this paper, we present a model-driven approach to the design of gamified applications, which accelerates the introduction of gamification elements in pre-existing or new applications. The approach relies on a data model of gamification features and on design patterns for the front-end, which encode the essential elements of gamification in a platform independent way.

Keywords: Model Driven Engineering, gamification, rapid prototyping, code generation, IFML.

1 Introduction

Gamification is defined as the injection of game elements in non-gaming applications [9]. Its main purpose is to boost the capacity of an application to engage users, improve their proficiency and satisfaction, and retain them. Gamification techniques have been applied in a variety of domains, both commercial and non-for-profit: customer relationship management [18], travel [33], education [25], fitness [38], and environmental awareness [16]. In

© 2020 *River Publishers. All rights reserved.*

its simplest form, gamification entails monitoring the *activity* of the user and her progress towards *goals* and providing *rewards* to her *achievements* [9]. All the four ingredients of gamification present themselves in an ample variety of forms: actions can be either endogenous to the application (e.g., accessing the application, creating content, executing tasks, etc) or external to it (e.g., specific behaviors detected through sensing and activity recognition, such as running, consuming less energy or water, visiting designated places, etc). They can be individual or collective, if team formation is exploited. Goals can be either set by the application (e.g., reaching a predefined level of expertise or of activity) or self-determined by the user (e.g., running a given amount of kilometers per month or saving 10% in energy or water consumption w.r.t. the preceding month). Achievements can be established by the application (e.g., by monitoring some progress indicators) or by other users (e.g., by collecting votes cast by the community or by expert users). Rewards can be intangible (e.g., promotion to a higher status in the application, assignment of points and badges) or real (e.g., goods, services, or price discounts). The achievement status of the user can be kept private or exposed, e.g., using public leaderboards. The above mentioned gamification ingredients are intertwined with the application functions and can evolve over time; the designer may start with a simple gamification scheme and add more advanced features progressively, or she may modify the gamification rules to steer the users' behavior towards a desired objective. Such a dynamic nature of gamification requires an agile development methodology, supporting the rapid prototyping of features and their adaptation over time. Agility can be attained by exploiting a *pattern-based model-driven* development life-cycle:

- *Design patterns* are defined as partial solutions to recurrent design problems [28]. They are particularly useful when some application features repeat, albeit with variations, across multiple domains. Their use can speed up development and also improve quality, because patterns embody design knowledge distilled in many solutions.

- *Model-Driven Engineering* advocates the use of models as the central artifacts of application development, from which the implementation can be derived [32]. Models are abstract representations of the application features, independent of technological details; as such, they pair well to the notion of patterns, because they allow the expression of the design knowledge embedded in patterns in a way that does not depend on the technical space in which a specific application instance is built.

In this paper, we define the model-driven patterns that embody the design knowledge necessary to develop a gamified application or to add gamification features to an existing system. The contribution of the paper can be summarized as follows:

- We recall the essential concepts of application gamification and define a reference architecture for gamified solutions. We formalize the gamification concepts and their relationships by means of a Gamification Domain Model.
- We identify a set of design patterns that embody the essential elements of application gamification; we represent such patterns using the Interaction Flow Modeling Language (IFML) [3] for the front end part, and UML sequence diagrams [31] for the back-end business logic.
- We showcase how the proposed model-driven pattern-based methodology has been applied to the development of two real world applications in the area of environmental awareness.

The rest of the paper is organized as follow: Section 2 surveys the related work in the areas of model-driven and pattern-based development, gamification and environmental awareness applications. Section 3 briefly describes the IFML modeling language. Section 4 defines the essential concepts of gamified solutions and provides a reference architecture for their development and execution. Section 5 presents the Domain Model supporting the design of gamification patterns and Section 6 presents the game rule engine and the exposed service API. Section 7 illustrates the patterns for the front-end of gamified applications, expressed in IFML. Section 8 discusses the use of the proposed model-driven pattern-based methodology in the development and evolution of two applications. Finally Section 9 provides the conclusions and highlights the envisioned future work.

2 Related work

Pattern-based Model-Driven Engineering. Model-driven engineering (MDE) is the systematic use of models as primary artifacts throughout the engineering life-cycle and is at the core of industrial tools such as Business Process Management Systems (BPMS) and Rapid Application Development (RAD) Platforms. Many sectors of the software development industry have adopted MDE, for example, to design SOA architectures [35], to secure SOA

data flows [17], to manage IoT infrastructures [6], and even to deploy machine learning processes on cloud infrastructures, e.g., with such tools as Azure Machine Learning Studio[1]. MDE and patterns match well: patterns capture design knowledge and models enable the reuse of such knowledge in a platform-independent way. Several Web MDE frameworks integrate patterns as building blocks for the automatic generation of user interfaces, as demonstrated e.g., in [29] and [10]. Koch et al. extended UML-based Web Engineering (UWE) to integrate patterns for Rich Internet Applications (RIAs), such as "auto-completion" or "periodic refresh" [20]; Fraternali et al. extended a web engineering methodology to represent the features of RIAs by allocating specific functionality in the appropriate tier and specifying suitable design patterns for dealing with the interaction between the tiers [12]. A pattern-based model-driven approach for safety-critical systems was proposed in [14], to model dependability patterns and enable their reuse across domains. Zdun et al. created an intermediate abstraction level consisting of pattern primitives, which can be used as building blocks for actual patterns applicable to the model-driven development of SOA processes [37].

Model-driven approaches for games and gamification. In this field, Herzing proposed the Gamification Modelling Language (GaML) [23], [15], a modelling language for game design formalized as a Xtext grammar[2]; the proposed approach transforms the textual definition of the game rules into JSON and Drools Rule Language (DRL) files, interpreted by software components based on the Unity[3] achievement system plugin and Drools Business Rule Engine[4] (BRE). Calderon et al. [4] proposed a graphical modelling language for gamification, supported by the Eclipse Modelling Framework[5] (EMF); the gamification domain, the rules and the interactions with non-gamified components can be defined graphically and the resulting model can be transformed into code for the Complex Event Engine (CEP) and for an Enterprise Service Bus (ESB). In both the above mentioned approaches a change of the gamification policies can only be accomplished in one of two ways: 1) by modifying the generated code, which requires expert programming skills to operate on generated code, which is not always human-readable; 2) by updating the model and regenerating the gamification artifacts, which requires the redeployment of the gamified components and makes evolution

[1] https://azure.microsoft.com/en-us/services/machine-learning-studio/
[2] https://www.eclipse.org/Xtext/
[3] https://unity.com/
[4] https://www.drools.org/
[5] https://www.eclipse.org/modeling/emf/

time consuming. The approach proposed in this paper factors out the gamification control rules from the gamification front-end patterns, which allows developers to change the former independently of the rest of the application.

Serious games and gamified environmental applications. Serious games and gamification have been applied in the environmental field because their motivational power is a desirable characteristic for engaging people in such areas as environmental education, consumption awareness and efficiency behaviours [24]. Ecogator [27] is an efficiency advisor mobile app that scans the energy labels of an appliance and provides hints about its efficiency, such as the annual running cost and the total cost over the product lifetime; it also compares two products to support the user in the decision-making and delivers energy efficiency tips on a daily basis. Drop! The question [11] is a card game with a digital extension for educating players about water saving; the game exploits a "push your luck" mechanics, in which the player repeatedly draws cards, with an increasing risk of losing; the cards are illustrated with water efficient and inefficient behaviours; when the player draws a "bad" card, she must scan a QR code on it with the mobile app and answer a water-related question. Other examples of applications providing environmental education are described in [22], [19] and [1]. Social Power [8] aims at raising energy consumption awareness for users in households and shared spaces, such as schools and libraries. It exploits social interactions and game mechanics to drive people towards more sustainable energy consumption; upon registration users are assigned to a team and receive individual and collective saving goals. The app monitors consumption, by connecting to smart meters, and assigns points to individuals and teams when they save. The SmartH2O project [30] focuses on water saving; its gamified web portal and mobile app connect to water smart meters and enable users to monitor their consumption in quasi-real-time and to pursue weekly water saving goals. enCOMPASS [13] is a project aimed at increasing awareness about efficient energy consumption; it uses smart meters and sensors to collect energy consumption, indoor climate and user activity information to provide personalized recommendations for energy saving based on the user profile, habits and preferred comfort level. The project exploits a gamified app and a hybrid (card and digital) game, to achieve the desired impact on the consumers. Some applications exploit virtual environments to teach efficient behaviours to users. An example is Water Mansion [36], a serious game in which users must execute daily tasks, such as showering or washing dishes; each action increases water consumption and reduces the "gold" that the user owns; the objective is to learn about water efficient consumption and its eco-

nomic impact. A similar approach is applied in EnerGAware [5], a mobile simulation game, in which the objective is to reduce the energy consumption of a virtual house with respect to the previous week. The player can execute actions, such as changing the location of the lamps in a room or turning off appliances (lights, TV, etc.). At the end of every week, the players receive points based on the energy saved. The SmartH2O and enCOMPASS projects, which have been developed with the methodology proposed in this paper, will be described in more detail in section 8.

3 Background: IFML in a nutshell

The Interaction Flow Modeling Language (IFML) [26] is an OMG standard for the platform-independent description of the front-ends of interactive applications. With IFML developers specify the organization of the interface, the content to be displayed, and the effect on the interface produced by the user interaction or by system events. The business logic of the actions activated by the user interaction can be modeled with any behavioral language, e.g., with UML sequence diagrams. Figure 1 shows the essential elements of the IFML metamodel.

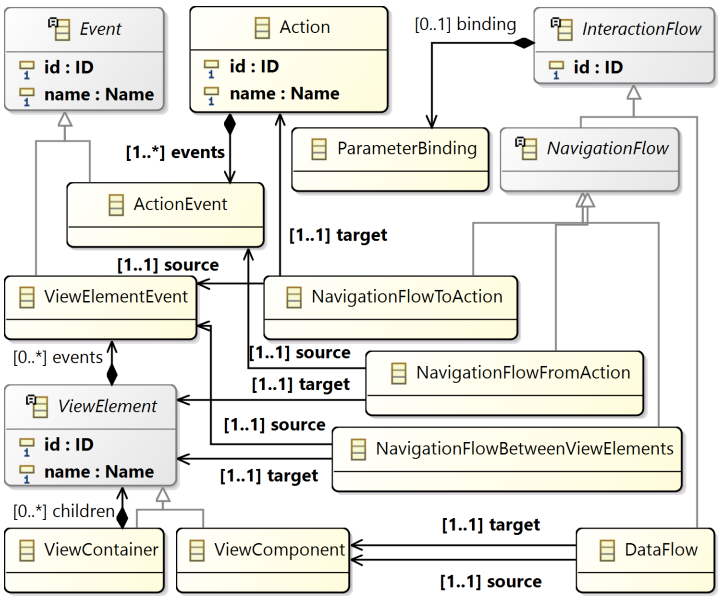


Figure 1: Essential elements of the IFML Metamodel

Interface Structure. The core IFML element for describing the front-end structure is the *ViewElement*, specialized into *ViewContainer* and *ViewComponent*. *ViewContainers* denote the modules that comprise the interface content; a ViewContainer can be internally structured in a hierarchy of subcontainers, e.g., to model a main interface window that contains several frames, which in turn contain nested panes, and so on. A ViewContainer can include *ViewComponents*, which represent the actual content of the interface. The generic ViewComponent specializes into different elements, such as lists, object details, data entry forms, and more. Figure 2 shows the notation: the Search ViewContainer comprises a MessageKeywordSearch *Form* ViewComponent, which represents a data entry form; MailBox includes a MessageList *List* ViewComponent, which denotes a list of items; finally, MessageViewer comprises a MessageContent *Details* ViewComponent, which displays one object. ViewComponents can have input and output parameters: a Details ViewComponent may have an input parameter that identifies the object to display, a form has output parameters corresponding to the submitted values, and a List ViewComponent has an output parameter that identifies the selected item.

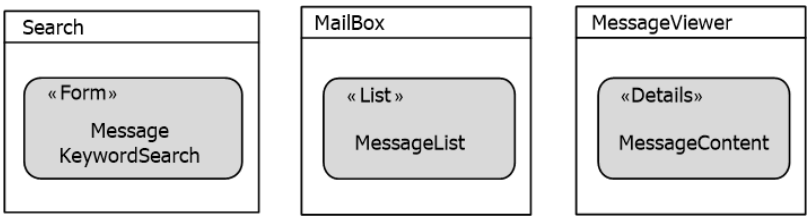
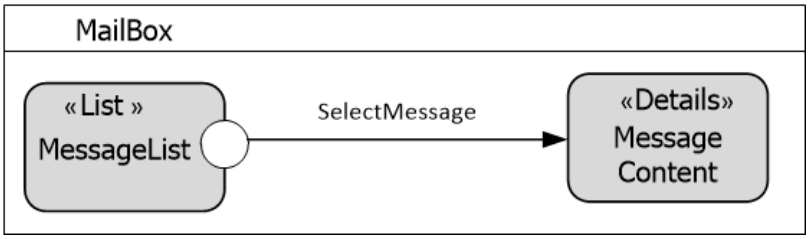


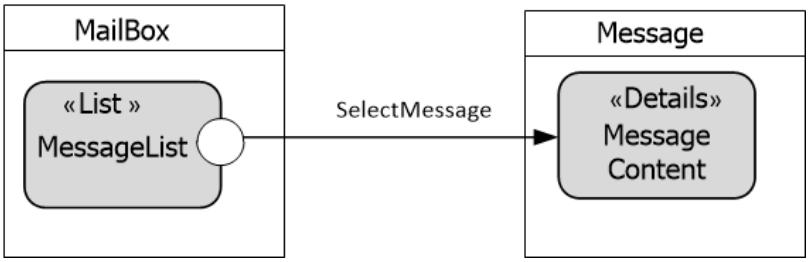
Figure 2: Example of ViewComponents within view containers

Events, Navigation and Data Flows. *ViewElements* (ViewContainers and ViewComponents) can be associated with *Events*, to express that they support the user interaction. For example, a List ViewComponent can be associated with an Event for selecting one or more items (as in Figure 3), and a Form ViewComponent with an Event for input submission. The effect of an Event is represented by a *NavigationFlow*, denoted by an arrow, which connects the Event to the ViewElement affected by it (as shown in Figure 3). When an event occurs, the target ViewElement of the NavigationFlow associated with it gets in view and the source ViewElement may stay in view or switch out of view, depending on the structure of the interface. In Figure 3a, the NavigationFlow associated with the SelectMessage Event

connects its source (MessageList, which displays a list of objects), and its target (MessageContent, which displays the data of an object). When the Event occurs, the content of the target ViewComponent is computed so to display the chosen object, and the source remains in view since it is in the same ViewContainer. In Figure 3b the source and target ViewComponents are in distinct ViewContainers (MailBox and Message); the SelectMessage Event causes the display of the Message ViewContainer, with its content, and the replacement of the MailBox ViewContainer, which exits from view.



(a) NavigationFlow between ViewComponent in the same ViewContainer



(b) NavigationFlow between ViewComponent in different ViewContainers

Figure 3: Example of NavigationFlow between ViewComponents

IFML can also show the objects from which ViewComponents derive content, their inputs and outputs, and the parameter passing from the source to the target of the NavigationFlow.

In Figure 4 the ViewComponents comprise a *DataBinding* element that identifies the data source, which can be an object class defined in the Domain Model of the application. Both the ViewComponents in Figure 4 derive their content from the MailMessage entity. MessageContent also comprises a *ConditionalExpression* i.e., a filter used to extract the content to publish; such ConditionalExpression is parametric: it extracts the object whose MessageID attribute value equals the Msg_ID parameter supplied by SelectMessage

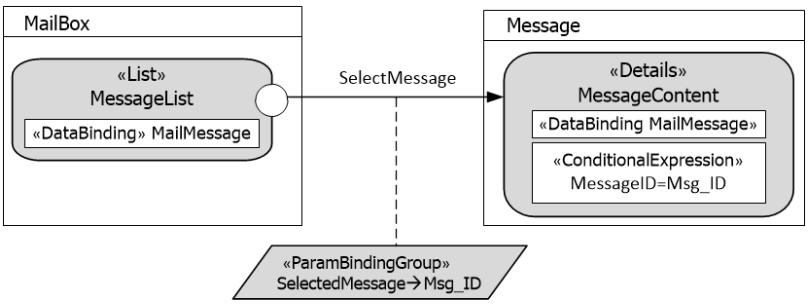


Figure 4: Example of DataBinding, ConditionalExpression, and ParameterBindingGroup

Event. The parameter passing rule is represented with a *ParameterBindingGroup* element associated with the Navigation Flow, which couples an output parameter of the source ViewComponent to an input parameter of the target ViewComponent. NavigationFlows enable the expression of the effect of an Event and the specification of parameter passing rules. Yet, a parameter passing rule can be expressed independently of an interaction Event, using *DataFlows*. Figure 5 shows the *DataFlow* construct, representing an input-output dependency between a source and a target ViewElement, denoted as a dashed arrow. MailViewer includes three ViewComponents: the MailMessages *List* is defined on the MailMessage entity, and shows a list of messages; the MessageContent *Details* is also defined on the MailMessage entity and displays the data of a message; the Attachments *List* is defined on the Attachment entity and shows a list of mail attachments. The identifier of the selected message is passed from MailMessages to MessageContent, which has a parametric *ConditionalExpression* to extract the message with the identifier provided in input. Also Attachments has a parametric ConditionalExpression, to select the attachments of the mail message provided in input. When the ViewContainer is accessed, the list of messages is displayed, which requires no input parameters. The DataFlow between MailMessages and MessageContent expresses a *parameter passing rule* between its source and target: even if the user does not trigger the Select Event, an object is randomly chosen from those displayed in the MailMessages *List* and supplied as input to MessageContent, which displays its data. Similarly, the DataFlow between the MessageContent and Attachments specifies an automatic parameter passing rule for the list of attachments. By triggering the Select event,

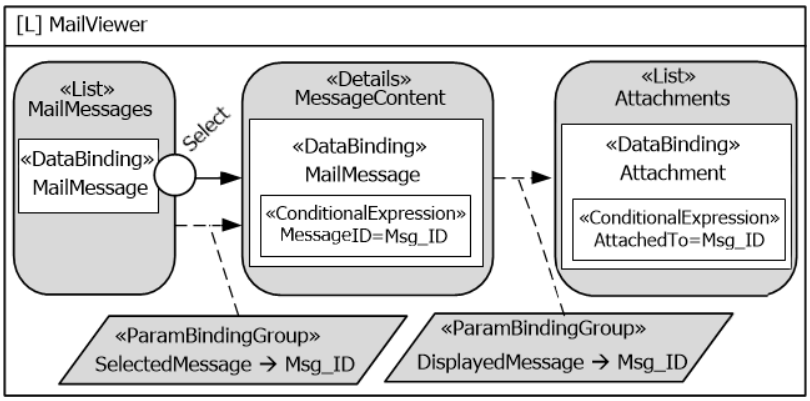


Figure 5: Example of DataFlows

the user can choose a specific message from the list and display its content and attachments.

Actions. An Event can also cause the triggering of business logic, executed prior to updating the state of the user interface; the IFML *Action* construct, represented by an hexagon symbol as shown in Figure 6, denotes the invoked program, which is treated as a black box, possibly exposing input and output parameters. The effect of an Event firing an Action and the possible parameter passing rules are represented by a NavigationFlow connecting the Event to the Action and possibly by DataFlows incoming to the Action from ViewElements of the interface. The execution of the Action may cause a change in the state of the interface and the production of input parameters for some ViewElements; this is denoted by termination events associated with the Action, connected by NavigationFlows to the ViewElements affected by the Action. Figure 6 shows an example of Action, for the creation of a new object. ProductCreation includes a *Form* with *SimpleField* sub-elements for specifying the data entry of a new product. The CreateNewProduct Event triggers the submission of the input and the execution of the CreateProduct Action. A ParameterBindingGroup is associated with the NavigationFlow from the CreateNewProduct Event, to express the parameter binding between the *Form* and the Action. The Action has two termination Events: normal termination lead to the visualization of the NewProductData ViewComponent within the NewProductDisplay ViewContainer; upon abnormal termination, an Event and NavigationFlow specify that an error message ViewComponent is displayed in a different ViewContainer.

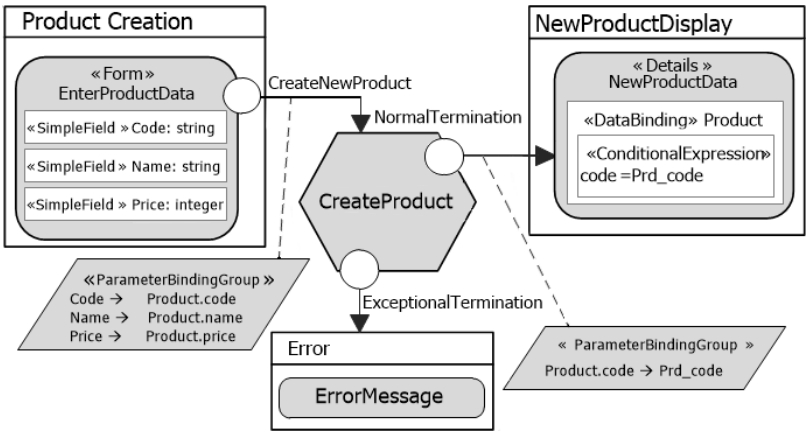


Figure 6: Example of Actions

4 Gamification Concepts and Architecture

Application gamification aims at engaging users by fostering their involvement and by enhancing their motivations to perform well in the accomplishment of a task [7]. Gamified platforms expose rules that guide the users through a progression of tasks and direct them towards the accomplishment of the defined objectives, while providing feedback and keeping them interested with elements that promote competition, collaboration and self-improvement. For example, fitness applications, such as Runtastic[6] and Nike Run Club[7], showcase the above mentioned gamification design principles: they motivate the users to achieve an objective, provide activity statistics, assign challenges (personal and collaborative) adequate to the user's level, award achievements for accomplished goals, promote competition through periodic leader boards and offer nutritional information to guide users. Before discussing the technical components of a gamified application, we introduce the concepts that characterize the design of gamification in a platform-independent and cross-domain way.

- **Action**: is an activity that the user can perform. Actions can be done within the platform (e.g., accessing the application, watching content, providing information or feedback, etc), or outside it; external actions

[6] https://www.runtastic.com
[7] https://www.nike.com/us/en_us/c/running/nike-run-club

are typically measured through sensors or activity recognition (e.g., running an amount of kilometres, consuming less energy, etc).

- **Goal**: is a user-defined or platform-defined measurable objective that involves performing a series of actions; goals are characterized by a target value to reach (e.g., a minimum rating of produced content, an amount of kilometers, a percentage reduction of water consumption, etc) and optionally by a target deadline when the objective will be verified.

- **Points**: are the "unit of merit" used to reward actions and to recognize the accomplishment of goals; they are sometimes called "credits", especially when they can be redeemed or exchanged for goods within or outside the platform.

- **Achievement**: is a recognition, typically a badge, assigned to the user when a certain level of progression in a specific area is reached. Achievements should be visible to other users in the community for social recognition and to promote competition.

- **Reward**: is a digital or real world item that users can claim when a predefined condition is fulfilled; it may require the user to exchange points for the reward, or it can be assigned when an achievement is attained without further requirements.

- **Leader board**: is a list of the players ordered by a merit criterion, such as collected points or completed activities. It may be computed immediately or periodically (weekly, monthly, etc). More than one leader board can be used: for example, a long-term leader board helps engaging expert users, whereas a short-term (e.g., weekly) one fosters the engagement of novices, who see their initial achievements publicly recognized.

- **Notifications**: are messages about important events or states, delivered periodically (e.g., at the end of established periods) or upon the occurrence of a condition (e.g., the attainment of an achievement). They help preserve motivation and direct the user's attention to topics or tasks of interest. Notifications can be delivered inside the application, or outside it, e.g. by email.

- **Thematic Areas**: are categories in which actions, goals, achievements, and rewards can be grouped. They are used when engagement relies on

a plurality of stimuli: for example, in a collaborative learning platform, thematic areas could span personal learning, collaboration with other users, reputation improvement, etc.

4.1 A Gamification Architecture

An architecture for gamified applications should minimize the interference of the gamification rules with the business logic of the application and limit the integration effort. Figure 7 presents the high-level architecture experimented in multiple gamification projects, described in Section 8, which proved effective in integrating gamification into web and mobile applications. Its modules embed the essential functions of a gamified application.

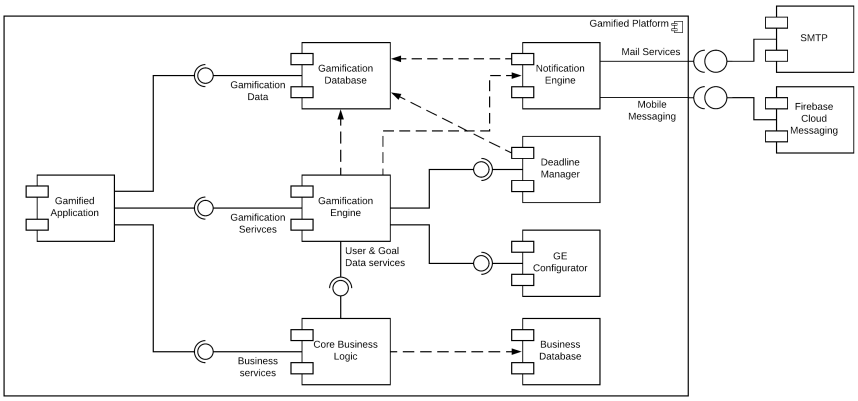


Figure 7: Proposed architecture for gamified applications.

The **Core Business Logic** module implements the actions to be performed by the user. Given the cross-domain nature of gamification, this module is represented as a generic software component, which delivers its services possibly relying on an application database storing the state of the non-gamified portion of the application. An important responsibility of this layer is to notify the Gamification Engine about the execution of actions by the user; the activities that are relevant for gamification should be registered in the Gamification Engine and their implementation should integrate the dispatching of relevant events to the Gamification Engine.

The **Gamification Engine** implements the registration of gamified actions, the establishment of goals, the assignment of points based on the

executed actions, the detection of achievements, and the delivery of rewards. The GE provides a service API for the Core business logic component to dispatch action events; it also exposes a query API for the gamification patterns embedded in the Gamified Application to retrieve information about the progress of users in the gamification exercise. Configurability is attained by factoring out the parameters that control gamification into a declarative specification, stored in the Gamification Database.

The **Gamification Database** stores the entities that allow the GE to execute the gamification rules and enable the Gamified Application to publish the user progress and state; it contains configuration parameters that enable the declarative specification of the gamification logic and facilitate its evolution. The domain model of the Gamification Database is discussed in Section 5.

The **Notification Engine** implements the logic for delivering the notifications that provide feedback to the users and remind them of their goals. Notifications can be configured to be triggered when the user perform actions, reach goals, attain achievements or unlock rewards. Notification delivery is controlled by parameters in the Gamification Database, to facilitate change. Configuration data also comprise the delivery channels and message templates; examples of delivery channels comprise email messages or messages of such systems as Google Firebase Messaging Framework[8].

The **Deadline Manager** is a chron process, which monitors the expiry of deadlines, configured in the Gamification Database, and calls the GE to enact the necessary procedures to check the status of the gamification and possibly notify users of relevant events.

The **Gamified Application** is the topmost layer of the architecture, which integrates the gamified view components into the business views. It exploits the Gamification Database, the gamification patterns, and the GE back-end services, to present game-related information contextualized in the business views and to capture and dispatch the user's action events.

The architecture of Figure 7 circumscribes the effort to integrate the business and the gamification logic within two components: 1) the Core Business Logic must be extended in such a way that the execution of business actions notifies the GE; no other modifications to the native business logic of the application are required; 2) the Gamified Application must complement the business user interface with the views and view components for disclosing the state of the gamification and engage the user in performing the gamified actions. It does so by incorporating the patterns illustrated in Section 7.

[8] https://firebase.google.com/docs/cloud-messaging/

The extension of the Core Business Logic to support the dialogue with the GE can be implemented with the help of the *Observer* pattern [28], whereby the GE acts as an Observer notified about the occurrence of events by the business logic, which acts as the *Subject*. The Observer-Subject relationship can be realized tightly, by extending the implementation of the business actions with explicit notification calls to the GE, or loosely, by having the GE poll the business logic component for status changes.

5 Gamification Domain Model

The Gamification Domain Model, shown in Figure 8, describes the entities and relationships used by the Gamification Engine to regulate the gamification actions, track user activity and progress, and assign achievements to the performing users. The provision of a Domain Model, independent of the code of the Gamification Engine and of the Gamified Application, facilitates the configuration of gamification and its dynamic evolution during the application maintenance, to adapt the engagement strategy to the evolution of the users' response to the gamification stimuli.

The main entities of the Gamification Domain Model are:

- **User**: is used for identification and profiling, with the usual information about the username, password, email, etc. If this information is already present in the application business database, the GE contains a replica or a view of the original data.
- **Gamification User**: specializes the User entity with attributes pertinent to gamification (total points, available credits, etc.).
- **Group**: is used to cluster users with different characteristics; it helps tailor the gamification stimuli to the specific needs of a user group and to compare users with similar characteristics, e.g., in the leader boards.
- **Thematic Area**: organizes actions and achievements that pertain to the same topic of interest, to focus the attention of the user and provide structure to her participation.
- **Action Type**: expresses a class of actions that can be performed, the configuration parameters that control the evaluation of such actions and the assignment of the points associated to them. The configuration parameters include the number of points awarded, the repeatability of the

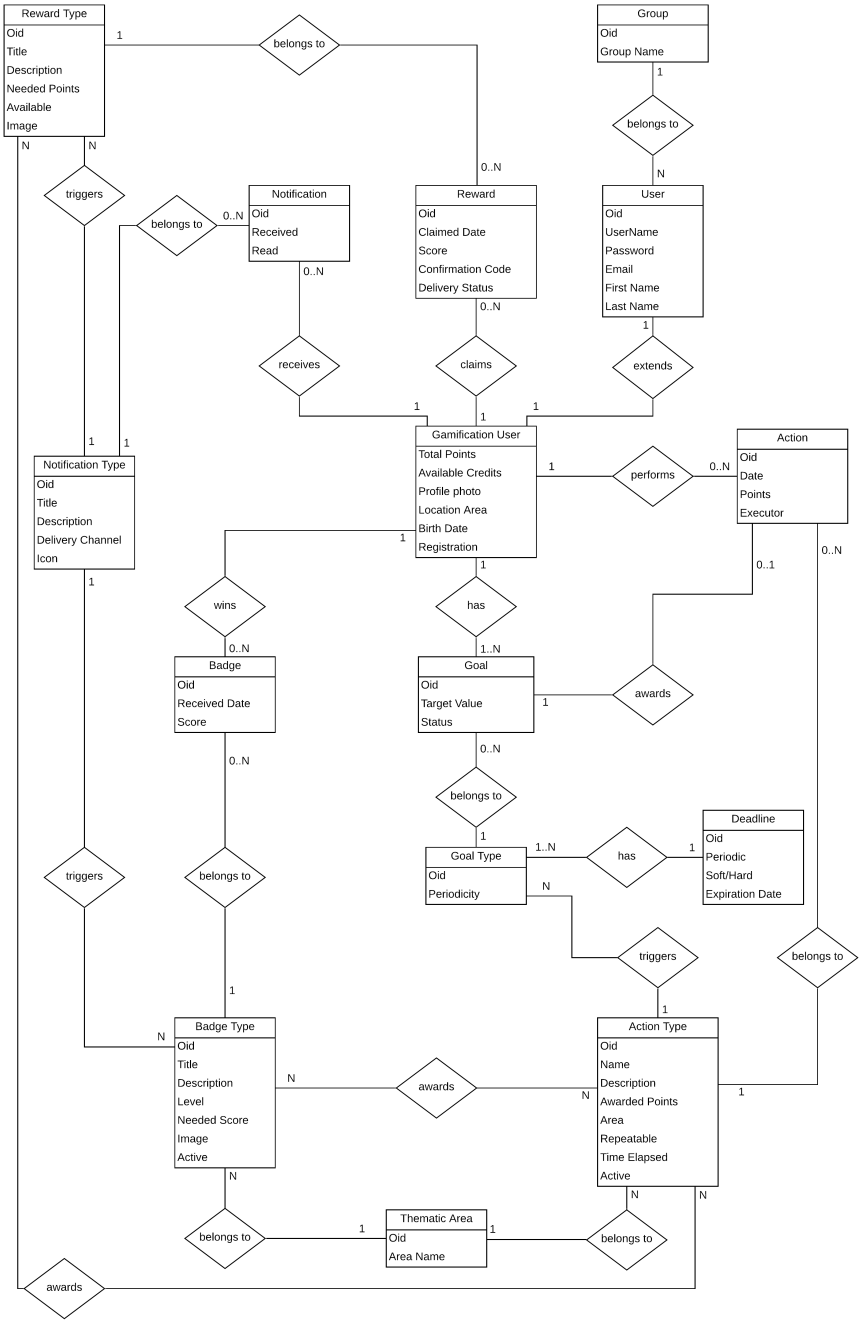


Figure 8: Gamification Domain Model

action, the minimum time interval between repeated executions, and the associated rewards. Action types can be associated to thematic areas.

- **Action**: denotes the actual instances of an action type performed by users.
- **Goal Type**: represents a category of goals. A goal type is associated with an indicator, which measures quantitatively the attainment of the goal. A goal type may be periodic or absolute: a periodic goal is checked at recurrent deadlines (e.g., every week or month), whereas an absolute goal is verified at a specific deadline (e.g, the end of the gamification exercise). A goal type is also associated with an action type, that denotes the action that must be fired to signal that the goal has been met and to update the gamification status of the user accordingly.
- **Goal**: represents the actual goals associated to the user; a goal is characterized by the user it belongs to, by a target value and by a status. The status can be in progress, achieved or missed. An achieved goal is associated to the action that has been created to trigger the assignment of points corresponding to its accomplishment.
- **Badge Type**: denotes a class of achievements, represented by badges that show the progression of the user in a thematic area. A badge type is characterized by a title, a description, a level, the required amount of points to attain it, the thematic area it belongs to, and an image that represent it visually.
- **Badge**: represents the actual badges acquired by the users.
- **Reward Type**: expresses a category of rewards, i.e., of prizes that a user can claim once she has performed the required actions or reached the required amount of points. Reward types have a title, a description, the required number of points, and an image.
- **Reward**: denotes the instances of the reward acquired by the users, characterized by a redemption date and by a confirmation code.
- **Deadline**: denotes a time point at which the status of the gamification should be checked; deadlines can be *periodic*, e.g, daily or monthly, or *absolute*, e.g., a fixed termination date of a gamification exercise. Deadlines can be *hard* or *soft*: hard deadlines are associated to goals, to

force their evaluation at specified times. Soft deadlines serve the purpose of checking the user's progress, with the aim of notifying the user and stimulating the attainment of goals.

- **Notification Type**: expresses a category of notifications, which can be sent to alert users about events and provide feedback. Notification types have a title, a description, an icon, and a delivery channel.
- **Notification**: denotes the actual notifications sent to the user.

6 Gamification Engine

The GE is the component that provides gamification services to applications. It exploits the Gamification Domain Model to manage user's activity, assigned points, verified goals, user's achievements, notifications, and rewards. The data that control the GE are edited with the GE Configurator, whereby the manager of the platform can create, modify, and remove gamification elements. The GE interacts with the *Notification Engine (NE)*, which delivers notifications. It can be seen as a process that handles two types of events: the posting of a user's action from the Gamified Application and the expiry of a gamification deadline, signalled by the Deadline Manager module.

6.1 Gamification Services

The GE exposes an API to process the user's actions and the expiry of deadlines, which we describe with UML sequence diagrams.

ProcessUserAction: this service takes in input the ID of a user, the ID of an action type, and the time stamp of the action occurrence; it checks the validity of the user's action, grants points, and verifies achievements and rewards. The process steps are as follows: 1. *ValidateUserAndAction*: if the action type and user are valid and active, then the action is assigned to the user; otherwise, it is ignored and the process ends. 2. *ValidateExecutability*: if the action type is non-repeatable and this is the first action of the type performed by the user, or if the action type is repeatable and the elapsed time since the last occurrence is larger than the interval configured for the action type, then the action is assigned to the user; otherwise, it is ignored and the process ends. 3. *GetPointsAndCheckAchievements*: the total number of points corresponding to the thematic areas of the action type is computed; if the required amount of points for an achievement in that area is reached, a badge is assigned to

the user and a corresponding notification request is sent to the Notification Engine. 4. *CheckRewards*: if the user has reached the necessary points, the reward is made claimable for the user and a signal is sent to the Notification Engine. 5. *UpdateUserPoints*: points are granted and leader boards updated.

The *ProcessUserAction* service is invoked by the Gamified Application, after a gamified action, and by the *CheckUserGoals* service, when the user has met a goal at a hard deadline. Figure 9 shows the sequence diagram of the *ProcessUserAction service*.

CheckUserGoals: takes in input a user ID and checks if the user has reached the goals associated with her. A goal is reached if a performance indicator, managed by the core business logic of the gamified application, is greater or equal to its target value. For example, in a technical support system the goal target value could represent the minimum number of "likes" received by the user's posts during the period. Figure 10 illustrates the processing steps of the service: 1. *RetrieveActiveGoals*: fetches the goals with "in progress" status and expired deadline. 2. *GetIndicatorValue*: The service queries the Core Business Logic component to retrieve the current value of the indicator needed to evaluate the goal. 3. *CheckGoalAccomplishment*: If the indicator value is greater or equal to the target value, the goal status is updated to "achieved", the *ProcessUserAction* service is called to award the corresponding points, and a goal accomplishment signal is sent to the Notification Engine. Otherwise, the goal status is updated to "missed" and a missed goal signal is sent to the Notification Engine.

The *CheckUserGoals* service is invoked by the DeadlineManager component when a hard deadline expires.

VerifyGoalProgress: this service is similar to *CheckUserGoal* but computes the percentage of the indicator value still missing to reach the goal, instead of verifying the goal completion. The *VerifyGoalProgress* service is invoked by the DeadlineManager component when a soft deadline expires.

Note that goal checking is executed asynchronously w.r.t. to the user's actions by the two services *CheckUserGoals* and *VerifyGoalProgress*. Both services are triggered by deadlines. The synchronous checking could be accomplished by extending the *ProcessUserActions* service with the logic to check the goal attainment, as done for the achievements.

RedeemRewards: this service takes in input a user ID and a reward ID and supports the redemption of rewards by the user. If the operation completes

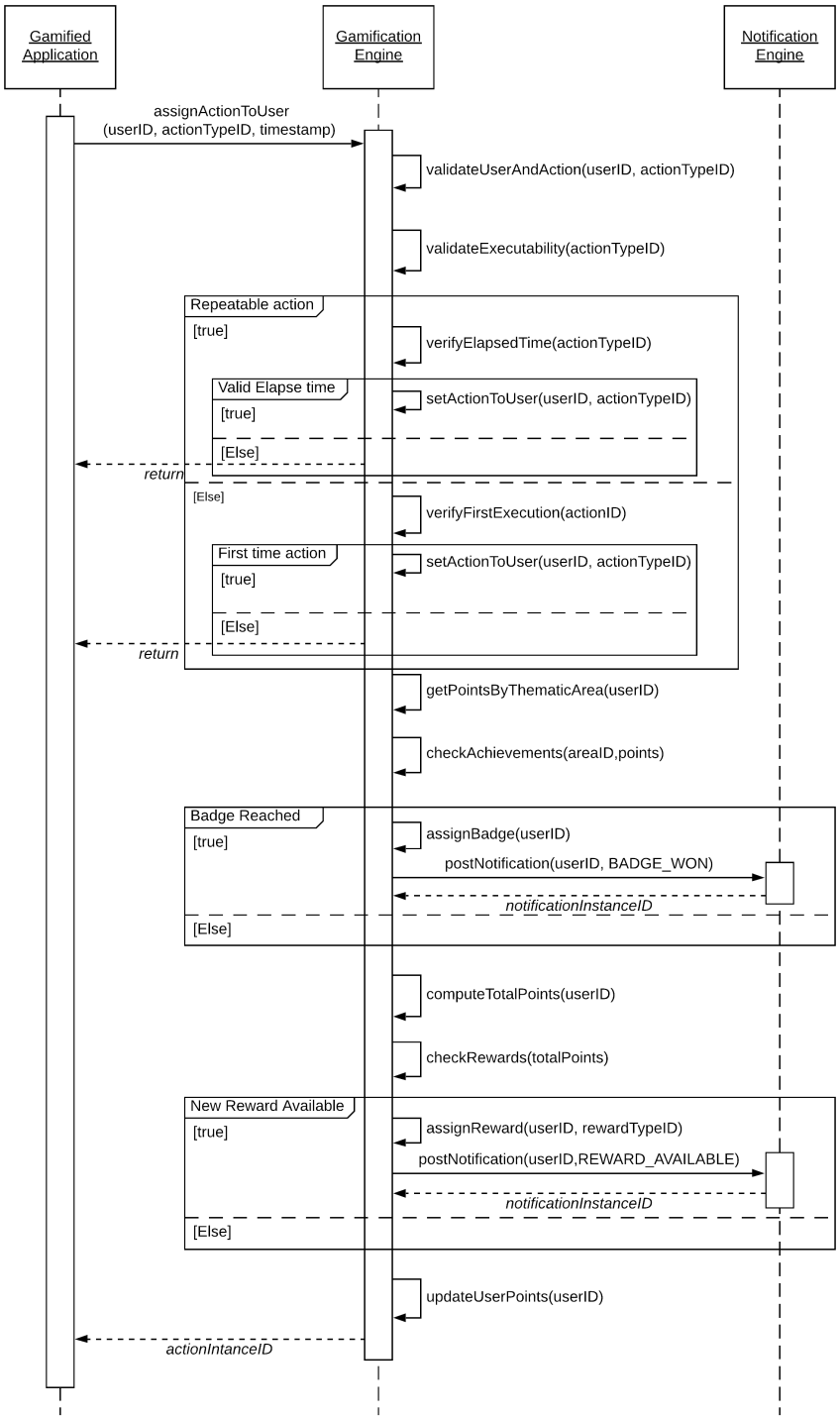


Figure 9: Sequence diagram of the ProcessUserAction service.

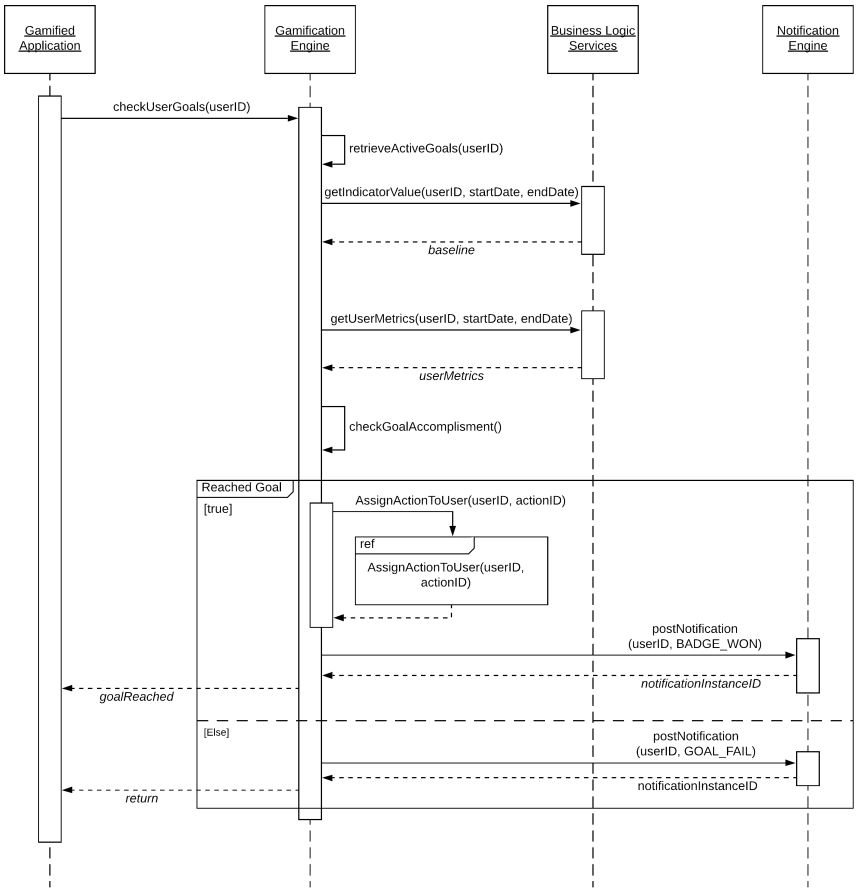


Figure 10: Sequence diagram of the CheckUserGoals service.

successfully, the user receives a confirmation code enabling the claim and the platform manager is notified about the event, so that he can handle the delivery process. Figure 11 shows the steps of the service.

1. *CheckCredit*: the user's credits and the availability of the reward are checked to confirm that the user can claim the item. If such requirements are not met, the service returns a failure message to the invoking Gamified Application. 2. *AssignReward*: if the requirements are met, the reward is assigned to the user, the credits are updated, a notification request including a reclamation code is sent to the NE.

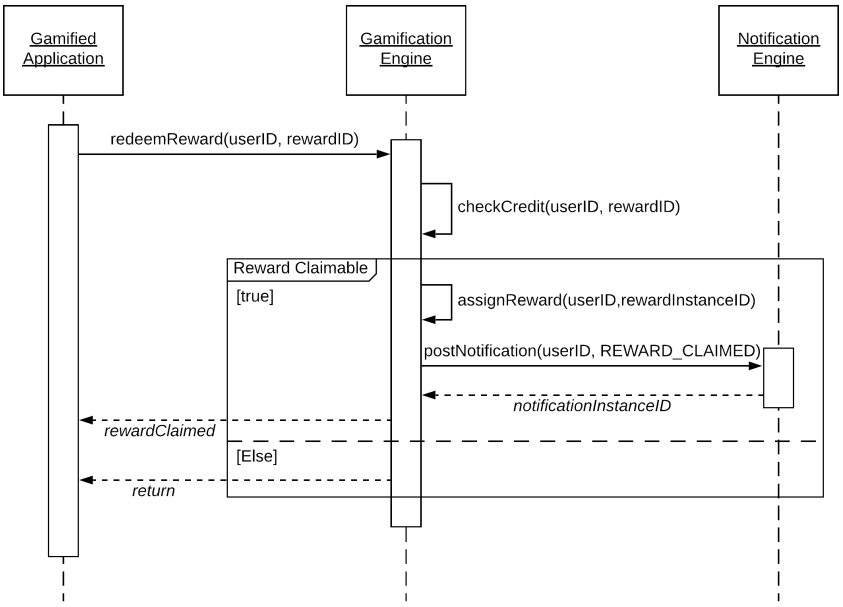


Figure 11: Sequence diagram of the RedeemRewards service.

The *RedeemRewards* service is invoked by the Gamified Application, whose interface allows the user to start the redemption process. Besides the above mentioned services, the GE also exposed CRUD operations for the management of the content of the Gamification Database (e.g., the creation of self-assigned goals by the users).

7 Gamification front-end patterns

Gamification impacts not only the back-end of a solution, but also the front-end, which must support the execution of the gamified actions and the display of the gamification status. Across the different domains in which gamification techniques can be applied, it is possible to recognize recurrent functions. In the spirit of MDE, such features can be captured as patterns, expressed by models that can be transformed into actual application components through model-to-text transformations. This section introduces the patterns that ex-

press the most common features of gamified applications, represented as IFML models following the notation explained in section 3.

7.1 Gamified Login and Home Page

The *Gamified Login and Home Page* pattern (shown in Figure 12) extends the well-known login functionality to award points when the user accesses the application, with the objective of encouraging continuous usage. The pattern consists of a ViewContainer (*Login*) comprising a form for inputting the user's credentials. The *Submit* event associated to the form triggers the *ValidateUserCredential* action, which checks the credentials provided by the user; in case of failure (event *AuthenticationFailed*), the *Login* ViewContainer is re-displayed, and shows the error message output by the *ValidateUserCredential* operation; in case of success, (event *AuthenticationSuccess*), the *ProcessUserAction* GE service is invoked, passing in input the current time stamp, the ID of the user, and the ID of the gamified action associated to the user's log in. Upon successful completion of the *ProcessUserAction*, a *Home* ViewContainer is displayed. The *Home* ViewContainer should include a reminder of the activities that the logged-in user can perform; these may be a mix of non gamified tasks and of gamified activities; to express this pattern in a general form, the model of the *Home* ViewContainer comprises two List ViewComponents, one for the non gamified and one for the gamified activities. In both ViewComponents an IFML ConditionalExpression (i.e., a predicate) is used to filter the operations and actions pertinent to the logged-in user.

A variant of the basic *Gamified Login and Home Page* pattern is obtained by extending the model of Figure 12 as follows: an event can be used to capture the failure of the *ProcessUserAction* operation and transfer the user to a gamification-specific error page, which contains a message warning the user of the reason why his action failed. This extension allows, for example, to manage malicious users that perform too frequent log in and log out operations with the intention of earning points and spamming the system. By setting a proper minimum interval for the log in gamified action it is possible to avert such an undesired behavior and warn the user of its consequences.

7.2 Gamified Action

The *Gamified Action* pattern, shown in Figure 13, demonstrates a generic way to gamify a task performed within the application. The pattern comprises a

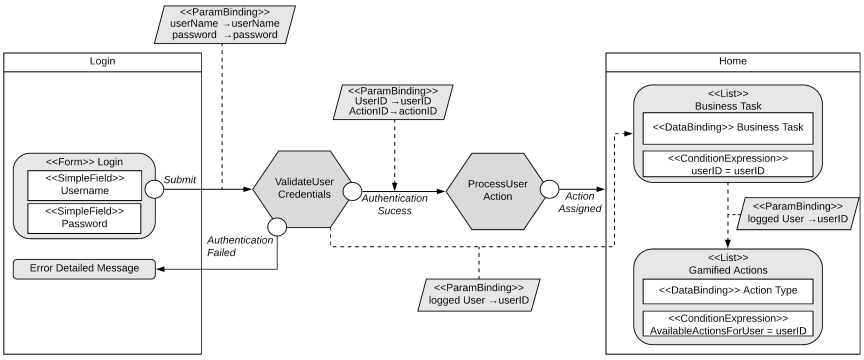


Figure 12: IFML pattern for gamified login and home page.

BusinessTask ViewContainer, which shows a list of available activities related to a gamified action. The *Select* event of the *PendingBusinessTasks* ViewComponent lets the user select the task to work on. Such a choice causes the display of a ViewContainer (generically named *CompleteBusinessTask* in Figure 13), whereby the user can perform the activity (e.g., by inputting data into a form). When the user finishes, she submits the task data to a core business action (generically named *ExecuteTask* in Figure 13) for validation and storage. If the business action completes successfully, then the GE *PerformUserAction* service is called to assign the action to the user. After the successful execution of the GE service, the *TaskCompleted* ViewContainer is displayed, which presents the details of the performed task. The *TaskCompleted* ViewContainer also includes a *ProgressInArea* List ViewComponent, which shows the badges for the thematic area related to the gamified action and provides immediate feedback about the user progress. Note that gamification "surrounds" the interface of the business activity: the *CompleteBusinessTask* ViewContainer does not contain gamification elements, to avoid distracting the user during the execution of the task.

The basic pattern can be extended by including a ViewComponent presenting the gamification status of the user in the thematic area of the gamified task also in the *BusinessTask* ViewContainer, to anticipate to the user the impact of executing an activity on her status.

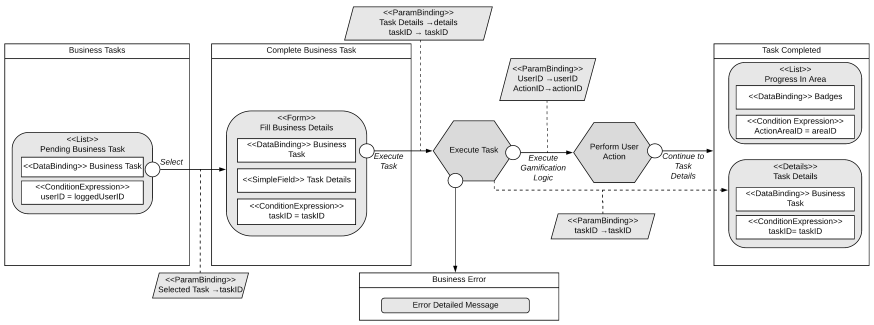


Figure 13: IFML pattern for Gamified Action.

7.3 Goal Selection and Progress

The *Goal Selection and Progress* pattern, shown in Figure 14, provides concise feedback about the user progress towards her goals, shows the status of the goals already established, and lets the user set her own self-assigned goals. The pattern comprises a *Goals* ViewContainer, in which a List ViewComponent enables the user to select the goal to visualize. The selection of a goal causes the display of the details of the chosen goal, which typically comprises the target value and the current value of the goal indicator. A Form ViewComponent (*SetSelfGoal*) enables the user to set a new goal, by inputting the goal type, a target value of the indicator and a deadline. When the user submits the data about the new self-established goal, the *SetUserGoal* action is triggered, which calls the GE *CreateGoal* service to update the Gamification Database. Upon the successful completion of the *SetUserGoal* action, the *Goals* ViewContainer is re-displayed, with the list of goals updated.

The basic version of the pattern can be enhanced by enriching the display of the current status of a goal with further information, e.g., a prediction of whether the current value of the indicator is such that the goal will be met by the deadline or else the user must increase her level of activity to attain the objective.

7.4 Gamified User Profile

The *Gamified User Profile* pattern, illustrated in Figure 15, shows a summary of the user status and progress in the gamification exercise. The pattern comprises a *User Profile* ViewContainer with three ViewComponents: a

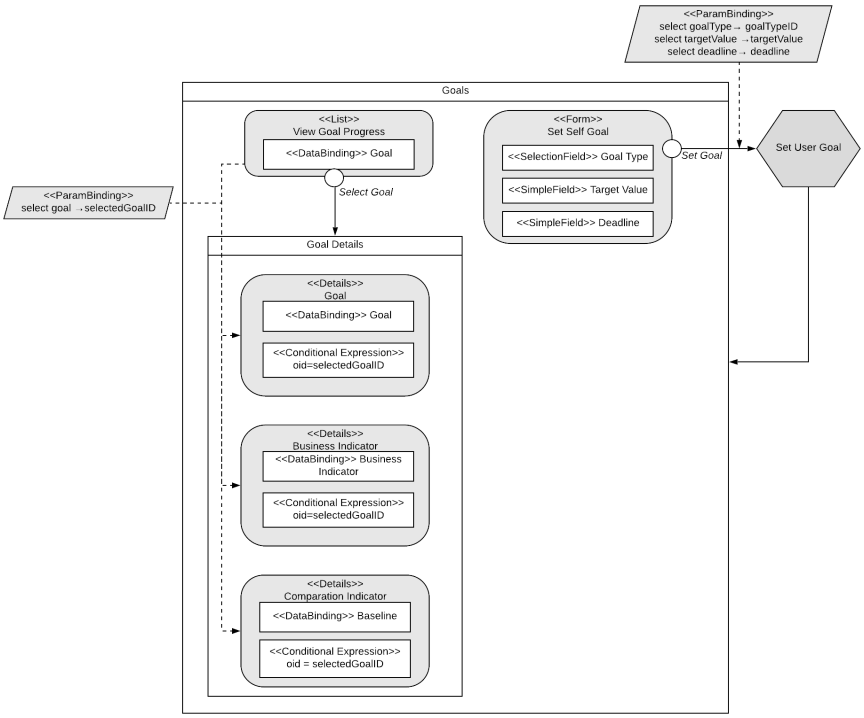


Figure 14: IFML pattern for Goal Selection and Progress.

UserDetails ViewComponent displays both general personal information, such as the user name and the profile photo, and gamification-specific data, such as the total points; a *Badges* List ViewComponent summarizes the acquired badges in the different thematic areas, and an *ActionHistory* List ViewComponent presents the actions performed by the user, in descending order of recentness. For each action, the execution time, the description of the action type and the granted points are displayed.

7.5 Leader Board

The *Leader Board* pattern consists of a Detail ViewComponent (*UserPointSummary*), showing the user points summary and personal information, and one or more List ViewComponents, displaying a ranked list of the users sorted by the selected performance criteria (points, badges, etc). The

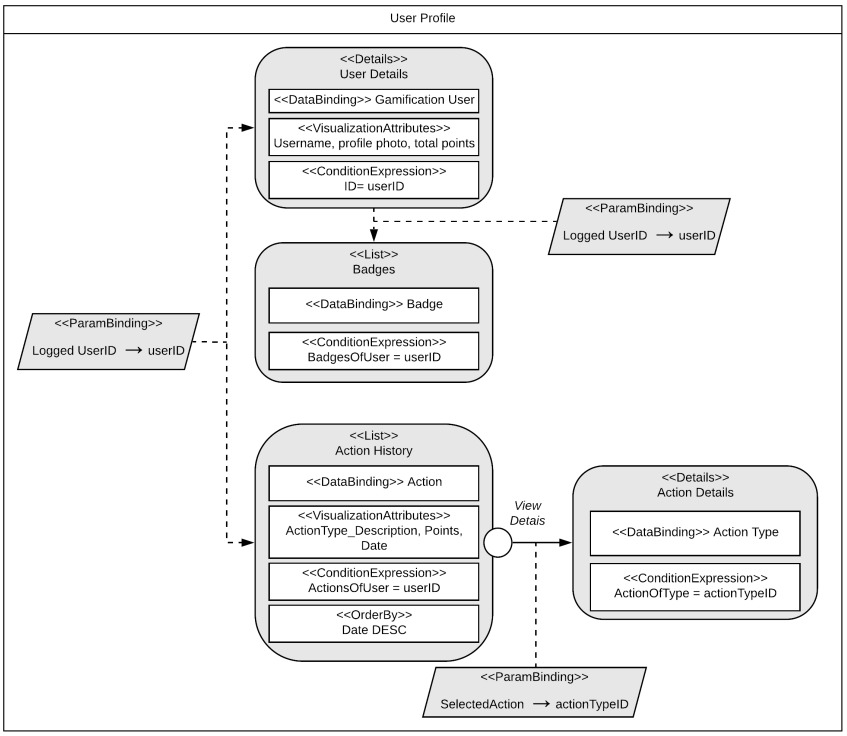


Figure 15: IFML pattern for Gamified User Profile.

exemplary pattern in Figure 16 includes two ranked lists. The first list (*PeriodicLeaderBoard*) shows the user performance in the current period (e.g., in the current week or month). The second list is an overall leader board that considers the entire duration of the gamification exercise. Both lists have a conditional expression that filters the users belonging to the same gamification group, so that users are compared with "competitors" with homogeneous characteristics.

7.6 Achievement Notification

The *Achievement Notification* pattern, shown in Figure 17, illustrates the interplay of notifications with the application views and the interaction of the user with the notifications. The pattern comprises a generic *BusinessOpera-*

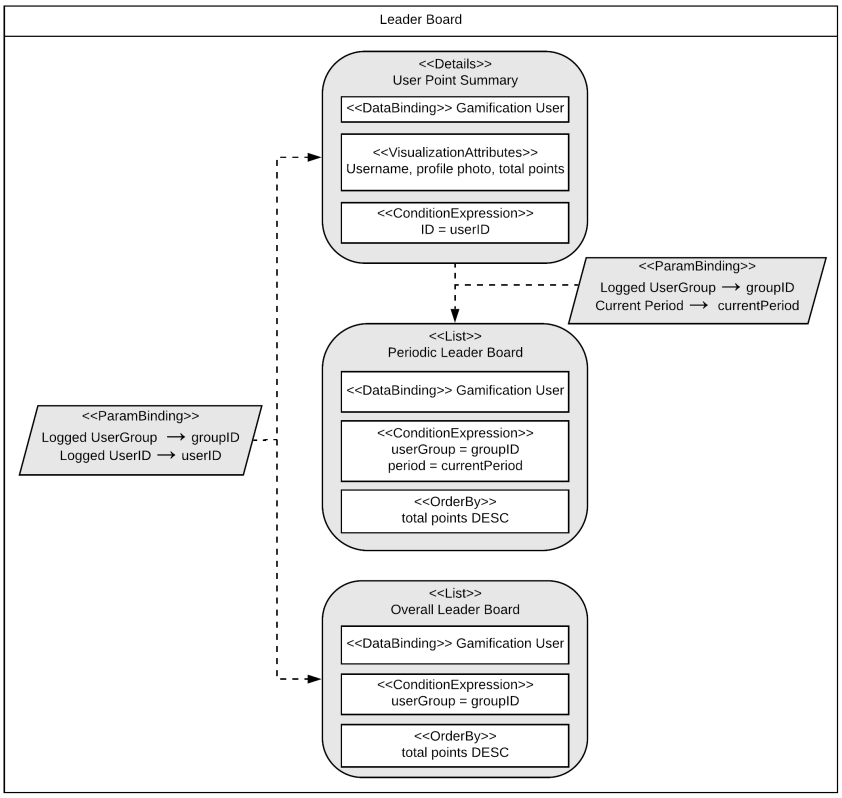


Figure 16: IFML pattern for Leader Board.

tions ViewContainer, which represents the interface whereby users perform the application activities, concisely represented by the *AvailableTasks* List ViewComponent; the ViewContainer also includes a details ViewComponent (*AchievementNotification*), which displays the notification signalled by the *AchievementAssigned* system event raised by the Notification Engine. The *AchievementNotification* ViewComponent shows only the title of the notification, to avoid cluttering the business view; however, the user can trigger the *ViewNotification* event to access a separate ViewContainer (*Notifications*) where she can inspect the whole content of the message; the ViewContainer comprises two ViewComponents: the *NotificationText* ViewComponent shows the full content of the current notification, including the

description of the achievement; the *AllNotifications* ViewComponent lists the received notifications, so that the user can inspect also the past messages. If the user wants to get more details about a current or past notified achievement, she can trigger the *ViewAchievementDetails* event, which causes the display of the *Badges* ViewContainer; this comprises a *BadgeDetails* ViewComponent showing the data (title, description, icon, and level) of the badge associated with the selected notification; the *Badges* ViewContainer also comprises an *AllBadges* List ViewComponent displaying all badges of the thematic area; badges already acquired by the user should be highlighted by the implementation of the *AllBadges* ViewComponent.

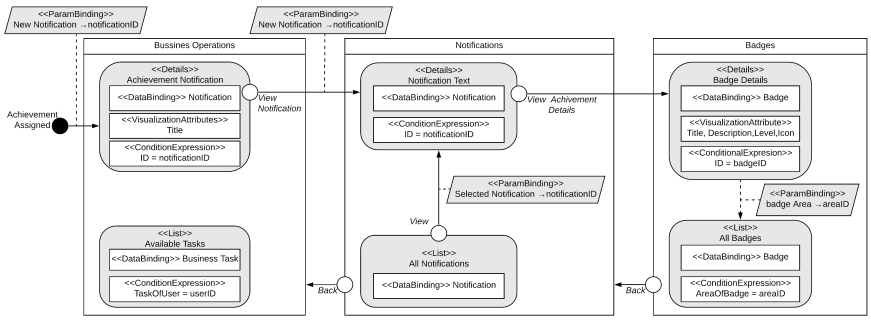


Figure 17: IFML pattern for Achievement notification.

The pattern can be extended in several ways: 1) including an event to dismiss a notification, so that it will no longer appear in the *AllNotifications* ViewComponents; 2) and adding to the *Badges* ViewContainer further components to see also the badges of other thematic areas.

7.7 Reward Visualization and Redemption

The *Reward Visualization and Redemption* pattern, shown in Figure 18, captures the user interaction for viewing the available rewards and for claiming one of them. The pattern comprises the *Credits* ViewContainer, which includes the *UserCredit* ViewComponent summarizing the user total points and the *AvailableRewards* ViewComponent showing the rewards that the user can claim. Triggering the *SelectReward* event, the user accesses the *RedeemReward* ViewContainer, which comprises a ViewComponent (*RewardDetails*) that displays the title, description and image of the selected reward. A Form

ViewComponent (*EnterShipmentData*) lets the users claim the reward by providing shipment details. When the user submits the form, the *RedeemReward* action is fired, which calls the corresponding GE service. After the successful completion of the *RedeemReward* action, the user is led to the *Confirmation* ViewContainer, where a ViewComponent presents the reward details and the confirmation code.

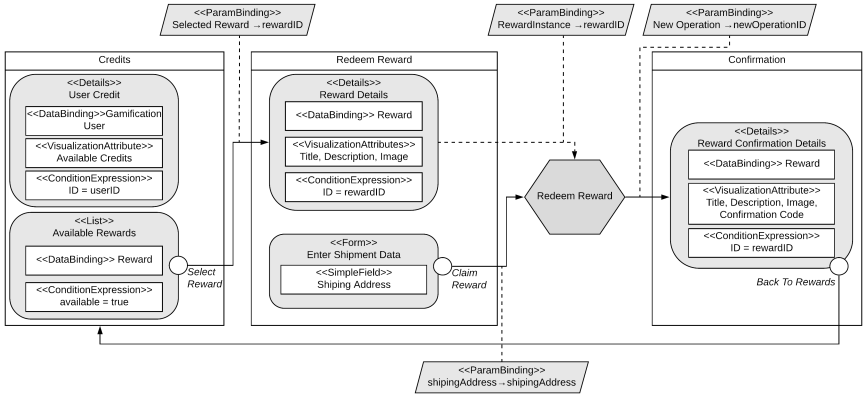


Figure 18: IFML pattern for Reward Visualization and Redemption.

The basic pattern can be extended by making the shipping status visible in the gamified application, so to keep the user engaged and avoid the need of providing shipment information through other channels, such as the email.

8 Case Studies

This section discusses the use of the proposed model-driven pattern-based methodology in the development and evolution of two real world applications in the area of environmental awareness: SmartH2O and enCOMPASS. The applications have been developed with WebRatio[9], a model-driven development environment based on IFML, which supports the definition of reusable modules for patterns and code generation for web and mobile platforms.

[9] https://www.webratio.com/

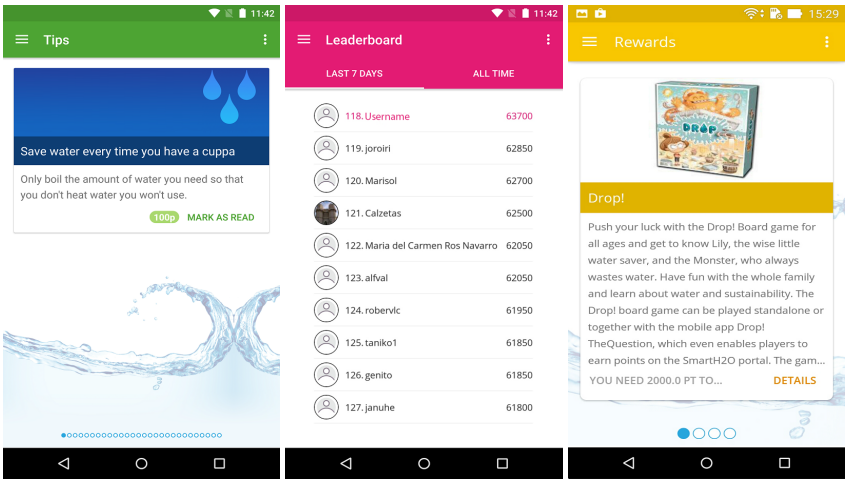
8.1 The SmartH2O project

SmartH2O is a project aimed at engaging consumers in water saving and at enabling water utilities to better manage water demand thanks to quasi-real time consumption data [30]. An ICT platform collects residential water smart meter data and a client application allows consumers to visualize their water consumption and to receive water saving tips (an example is shown in Figure 19a). The SmartH2O client application exploits gamification to motivate users to change their water consumption behaviour using virtual, physical, and social incentives. The gamified application assigns points to each user access (pattern *Gamified login and Home page*) and to a variety of actions (pattern *Gamified Action*), including filling-in profile information, reading tips, watching videos, sharing tips on social networks and inviting friends. Users can check platform-assigned goals and set their own objectives (pattern *Goal Selection and Progress*). Weekly gamification deadlines are established and two leader boards are used: weekly and overall. Figure 19b shows the interface of the pattern *Leader Board*. The top-3 users are notified via email of their achievement (pattern *Achievement notification*) and receive prizes, such as water-related board games, tickets for museums, and gift cards. Figure 19c illustrates the interface of the pattern *Reward Visualization and Redemption*. Finally, users can monitor their progress through a profile widget (pattern *Gamified User Profile*), which summarizes the water consumed in the period, the obtained points, the acquired badges and the executed actions.

SmartH2O was deployed in a small municipality in Cantone Ticino in Switzerland, and in Valencia, a large urban centre in Spain. Thanks to its use, an average reduction in consumption of 10% in Switzerland and of 20% in Spain has been observed [34]. After the end of the project, the participants kept the water saving habits, which provides evidence of the long-lasting behavioral change effects that a gamified platform can have over a community.

8.2 The enCOMPASS project

enCOMPASS is an ongoing project that implements a socio-technical approach to behavioural change for energy saving. It develops innovative tools to make energy consumption data understandable to different type of users, from residential consumers and school pupils to utility managers, empowering them to collaborate in order to achieve energy savings and manage their energy needs in efficient, cost-effective and comfort-preserving ways [13]. Smart meters and sensors collect energy consumption data and indoor in-

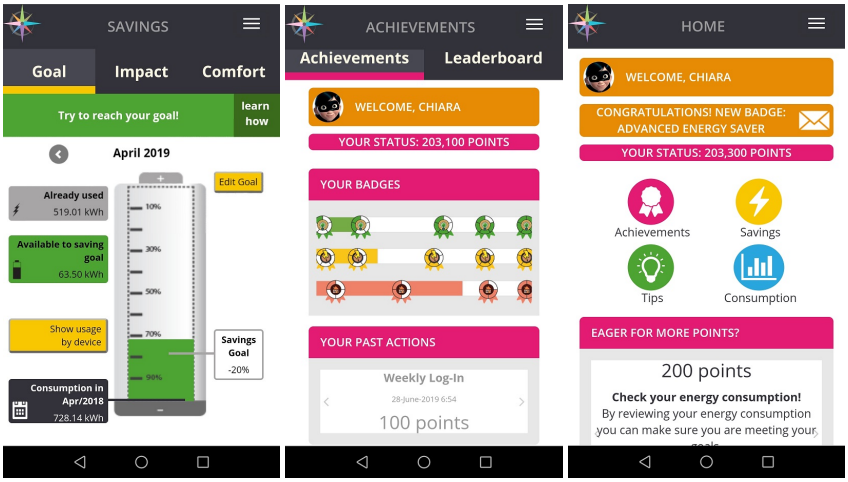


(a) (b) (c)

Figure 19: SmartH2O Views representing the patterns: a) Gamified Action, b) Leader Board, c) Reward Visualization and Redemption

dicators, such as temperature, humidity, and luminosity. Data are analyzed to infer the user activity and comfort standards and to provide personalized energy saving recommendations, based on the user's profile, habits, and preferred comfort level. A mobile application lets users explore consumption data under multiple visualizations, the indoor climate and comfort indicators, and personalized energy saving recommendations. Gamification is exploited to improve engagement and motivate the users to provide feedback about the personalized recommendations and their comfort levels. The gamification elements are divided into three thematic areas: learning, saving and profiling. The energy saving area encourages users to establish a saving goal at the beginning of every month. Figure 20a shows the realization of the *Goal Selection and Progress* pattern. A battery metaphor represents the goal indicator value, i.e., the amount of energy already consumed in the month, and the distance between the current value and the goal target. Users that reach the saving goal receive points proportional to the reduction goal. An *Achievements* page, shown in Figure 20b, realizes the *Gamified User Profile* pattern and lets the user check their progress in the thematic areas and browse their action history. In-app notifications and mobile alerts are regularly sent to notify users about the important events in the platform. Figure 20c shows the

implementation of the *Achievement Notification* pattern in the Home page of the app. The enCOMPASS client application implements all the other patterns introduced in Section 7, applies gamification to a broad spectrum of actions, and provides a rich set of views, which blend the display of business and gamification data.



(a) (b) (c)

Figure 20: enCOMPASS Views representing the patterns a)Goal Selection and Progress, b)Gamified User Profile, and c)Achievement notification

enCOMPASS is currently used by households, schools, and public buildings in three sites in Switzerland, Germany and Greece. The first analysis reveals a 10 to 12 % consumption reduction for the residential consumers [21]. A complete analysis is planned at the end of the project, to understand the overall effect of the intervention in households, public buildings and schools.

8.3 Discussion

Gamification is an excellent case for pattern-based model driven development, because: 1) it relies on well-defined functions that apply, with variations, across all application domains; 2) it requires quick evolution, to adapt the gamification rules to the users behavior; 3) it intersects all the tiers of an application and integrates within multiple views of the business inter-

face. Expressing gamification patterns in a platform-independent modeling language, such as IFML, provides several benefits: 1) it allows developers to focus on the core elements of the gamification (which actions to gamify, what rules to establish for controlling the execution and rewarding of actions, how to blend business and gamification views) in a high level way, deferring lower level, yet fundamental, aspects such as the visualization of the patterns to the later stage of code generation; 2) it allows design decisions about gamification to be factored out of the application source code, facilitating the evolution of the same application and the porting of design decision from one application to another. These benefits, which are generally useful for all systems, are essentials for user-centric gamified applications, where the main objective is engagement and retention. The ability to change the gamification features quickly allows the fast implementation of such critical updates as the addition of new engagement stimuli, more effective visualizations of the user's progress, and countermeasures to avert undesired behavior.

8.3.1 Analysis of the case studies

The SmartH2O and enCOMPASS experience demonstrated the usefulness of an application-agnostic gamification architecture and of model-driven gamification patterns in several stages of the development and maintenance process.

The principal lessons learned from the application of the proposed approach can be summarized as follows:

- The use of a formal Domain Model helped align the terminology and concepts across heterogeneous stakeholders and reason on the nuances of gamification early in the project. The distinction among actions, goals, and achievements (either rewards or badges) and the classification of the different types of progress monitoring deadlines helped framing the requirements quickly. The Domain Model concepts generalized well from the SmartH20 project to the enCOMPASS project, despite the greater complexity of the latter in terms of collected data, types of users, gamification rules, and visualization requirements.

- The availability of a "catalog" of front-end patterns helped reduce the space of the possible interface designs to a manageable size, which in turn enabled the rapid convergence to an accepted application configuration. Front-end patterns were mocked-up and different assemblies were discussed during the prototyping phase, speeding up consensus.

- The *Gamified Action* pattern proved the most useful, as it embodies the essence of gamification, which is the capture of specific users' actions that should be tracked and rewarded. The pattern distinguishes the application-dependent parts (e.g., the GUI for accomplishing the specific gamified task) and the application-agnostic parts (the tripartite structure selection-execution-confirmation and the signaling of the action to the Gamification Engine). Its use helped regularize the application design across very different tasks.

- The *Goal Selection and Progress* pattern proved the most complex to apply. The high-level nature of the pattern, which speaks in terms of generic business indicators, baselines and progress visualization, required a rather intense domain-specific customization to embed it into the concrete application. This may prompt for the identification of sub-patterns, adapted to less generic cases (e.g., distinguishing automatically assigned and self-set goals, periodic and non-periodic goal checking, different forms of progress prediction and visualization, etc.).

- The adoption of a pattern-based MDE approach shifted most effort to the presentation customization phase, which had to be realized manually by implementing ad hoc presentation templates applied to the IFML patterns during code generation. This is a well-known problem of MDE in general, which remains the bottleneck also in gamified, pattern-based applications. We are working on methods to simplify the integration of handwritten and automatically generated code to alleviate the burden of customizing the visualization of presentation-agnostic design patterns [2].

As a final remark, we note that the cross-domain nature of model-driven gamification patterns is further supported by the fact that the design schemes discussed in Section 7 and employed in SmartH2O and enCOMPASS are the same employed in a quite distinct gamification project, the technical support community of WebRatio[10], where the business data, the business views, and the target users are extremely different.

[10] https://www.webratio.com/community

9 Conclusions and future work

The paper describes a pattern-based model-driven methodology for the development of gamified applications, which expresses the gamification concepts within a Domain Model, a run-time architecture and a set of patterns, to facilitate the integration of gamification into existing or new applications. The identified patterns embody recurrent features of gamified application, which are synthesized into IFML models that promote reuse and customization through Mode-Driven Engineering techniques. The proposed approach was put to work in two real-world scenarios, showing that the model-driven encoding of gamification patterns promotes the reuse of design knowledge independently of the technology domain (web, mobile web and native mobile) and across applications. Factoring gamification rules out of the code in a gamification data model enabled the fast (re)configuration of the gamification engine and eased the adaptation to different scenarios. In the future, the proposed methodology can be extended to cover other aspects not addressed by IFML, such as the visualization patterns of the gamification elements. Presentation-oriented models could be devised to capture domain-specific specializations of the IFML components, such as *Goal Status* or *Gamified Action Widget*, which incorporate the knowledge on how to present gamification elements present in IFML patterns in a language-independent way. The Gamification Engine can also be extended by adding a data analysis component for the automatic detection and correction of undesired behaviours, such as spamming, or for the automatic adaptation of rules and points based on the user activity.

Acknowledgements This work is partially supported by the "enCOMPASS - Collaborative Recommendations and Adaptive Control for Personalised Energy Saving" project funded by the EU H2020 Programme, grant agreement no. 723059.

References

[1] Ronita Bardhan, Chaitra Bahuman, Imrankhan Pathan, and Krithi Ramamritham. Designing a game based persuasive technology to promote pro-environmental behaviour (peb). In *2015 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 1–8. IEEE, 2015.

[2] Carlo Bernaschina, Emanuele Falzone, Piero Fraternali, and Sergio Luis Herrera Gonzalez. The virtual developer: Integrating code generation and manual development with conflict resolution. *ACM Trans. Softw. Eng. Methodol.*, 28(4):20:1–20:38, September 2019.

[3] Marco Brambilla and Piero Fraternali. *Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML.* Morgan Kaufmann, 2014.

[4] Alejandro Calderón, Juan Boubeta-Puig, and Mercedes Ruiz. Medit4cep-gam: A model-driven approach for user-friendly gamification design, monitoring and code generation in cep-based systems. *Information and Software Technology*, 95:238–264, 2018.

[5] Miquel Casals, Marta Gangolells, Marcel Macarulla, Alba Fuertes, Vincent Vimont, and Luis Miguel Pinho. A serious game enhancing social tenants' behavioral change towards energy efficiency. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[6] F. Ciccozzi, I. Crnkovic, D. Di Ruscio, I. Malavolta, P. Pelliccione, and R. Spalazzese. Model-driven engineering for mission-critical iot systems. *IEEE Software*, 34(1):46–53, Jan 2017.

[7] Mihaly Csikszentmihalyi. Flow and the psychology of discovery and invention. *HarperPerennial, New York*, 39, 1997.

[8] Vanessa De Luca and Roberta Castri. The social power game: A smart application for sharing energy-saving behaviours in the city. *FSEA 2014*, 27, 2014.

[9] 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, Tampere, Finland, September 28-30, 2011*, pages 9–15, 2011.

[10] Jürgen Engel, Christian Märtin, and Peter Forbrig. A concerted model-driven and pattern-based framework for developing user interfaces of interactive ubiquitous applications. In *LMIS@ EICS*, pages 35–41, 2015.

[11] Piero Fraternali, Giorgia Baroffio, Chiara Pasini, Luca Galli, Isabel Micheel, Jasminko Novak, and A Rizzoli. Integrating real and digital games with data analytics for water consumption behavioral change: a demo. In *Proceedings of the 8th International Conference on Utility and Cloud Computing*, pages 408–409. IEEE Press, 2015.

[12] Piero Fraternali, Sara Comai, Alessandro Bozzon, and Giovanni Toffetti Carughi. Engineering rich internet applications with a model-driven approach. *ACM Transactions on the Web (TWEB)*, 4(2):7, 2010.

[13] Piero Fraternali, Sergio Herrera, Jasminko Novak, Mark Melenhorst, Dimitrios Tzovaras, Stelios Krinidis, Andrea Emilio Rizzoli, Cristina Rottondi, and Francesca Cellina. encompass—an integrative approach to behavioural change for energy saving. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[14] Brahim Hamid and Jon Perez. Supporting pattern-based dependability engineering via model-driven development: Approach, tool-support and empirical validation. *Journal of Systems and Software*, 122:239–273, 2016.

[15] P. Herzig, K. Jugel, C. Momm, M. Ameling, and A. Schill. Gaml - a modeling language for gamification. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 494–499, Dec 2013.

[16] Tad Hirsch. Water wars: designing a civic game about water scarcity. In *Proceedings of the Conference on Designing Interactive Systems, Aarhus, Denmark, August 16-20, 2010*, pages 340–343, 2010.

[17] Bernhard Hoisl, Stefan Sobernig, and Mark Strembeck. Modeling and enforcing secure object flows in process-driven soas: an integrated model-driven approach. *Software and System Modeling*, 13(2):513–548, 2014.

[18] Kai Huotari and Juho Hamari. Defining gamification: a service marketing perspective. In *International Conference on Media of the Future, Academic MindTrek '12, Tampere, Finland, October 3-5, 2012*, pages 17–22, 2012.

[19] Erik Knol and Peter W De Vries. Enercities-a serious game to stimulate sustainability and energy conservation: Preliminary results. *eLearning Papers*, (25), 2011.

[20] Nora Koch, Matthias Pigerl, Gefei Zhang, and Tatiana Morozova. Patterns for the model-based development of rias. In *International Conference on Web Engineering*, pages 283–291. Springer, 2009.

[21] Ksenia Koroleva, Mark Melenhorst, Jasminko Novak, Sergio Gonzalez, Piero Fraternali, and Andrea-Emilio Rizzoli. Designing an integrated socio-technical behaviour change system for energy saving. *Energy Informatics*, 2, 09 2019.

[22] George E Lee, Yongwen Xu, Robert S Brewer, and Philip M Johnson. Makahiki: An open source game engine for energy education and conservation. *Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii*, 96822:11–07, 2012.

[23] Amir Matallaoui, Philipp Herzig, and Rüdiger Zarnekow. Model-driven serious game development integration of the gamification modeling language gaml with unity. In *2015 48th Hawaii International Conference on System Sciences*, pages 643–651. IEEE, 2015.

[24] Luca Morganti, Federica Pallavicini, Elena Cadel, Antonio Candelieri, Francesco Archetti, and Fabrizia Mantovani. Gaming for earth: Serious games and gamification to engage consumers in pro-environmental behaviours for energy efficiency. *Energy Research & Social Science*, 29:95–102, 2017.

[25] Siobhan O'Donovan, James E. Gain, and Patrick Marais. A case study in the gamification of a university-level games development course. In *2013 South African Institute for Computer Scientists and Information Technologists, SAICSIT '13, East London, South Africa, October 7-9, 2013*, pages 242–251, 2013.

[26] OMG. Interaction flow modeling language (IFML), version 1.0. http://www.omg.org/spec/IFML/1.0/, 2015.

[27] Melanie Peham, Gert Breitfuss, and Rafael Michalczuk. The ecogator app: gamification for enhanced energy efficiency in europe. In *Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality*, pages 179–183. ACM, 2014.

[28] Wolfgang Pree and Erich Gamma. *Design patterns for object-oriented software development*, volume 183. Addison-wesley Reading, MA, 1995.

[29] Frank Radeke, Peter Forbrig, Ahmed Seffah, and Daniel Sinnig. Pim tool: support for pattern-driven and model-based ui development. In *International Workshop on Task Models and Diagrams for User Interface Design*, pages 82–96. Springer, 2006.

[30] AE Rizzoli, A Castelletti, A Cominola, P Fraternali, A Diniz dos Santos, B Storni, R Wissmann-Alves, M Bertocchi, J Novak, and I Micheal. The smarth2o project and the role of social computing in promoting efficient residential water use: a first analysis. 2014.

[31] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified modeling language reference manual, the*. Pearson Higher Education, 2004.

[32] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[33] Marianna Sigala. The application and impact of gamification funware on trip planning and experiences: the case of tripadvisor's funware. *Electronic Markets*, 25, 01 2015.

[34] SmartH20 Consorsium. Deliverable 7.2 validation report. http://smarth2o.deib.polimi.it/wp-content/uploads/2017/03/sh2o_D7.2_SES_WP7_validation_report_v1.1.pdf, 2016.

[35] Jihed Touzi, Fréderick Benaben, Hervé Pingaud, and Jean Pierre Lorré. A model-driven approach for collaborative service-oriented architecture design. *International journal of production economics*, 121(1):5–20, 2009.

[36] UrbanWater. Deliverable 5.6 – game solution for customer empowerment using water consumption data. http://urbanwater-ict.eu/wp-content/uploads/2015/12/UrbanWater-D5.6_v.F.pdf, 2015.

[37] Uwe Zdun and Schahram Dustdar. Model-driven and pattern-based integration of process-driven soa models. *International Journal of Business Process Integration and Management (IJBPIM)*, 2(2):109–119, 2006.

[38] Oren Zuckerman and Ayelet Gal-Oz. Deconstructing gamification: evaluating the effectiveness of continuous measurement, virtual rewards, and social comparison for promoting physical activity. *Personal and Ubiquitous Computing*, 18(7):1705–1719, 2014.