A Platform-Independent Model for Videogame Gameplay Specification

Emanuel Montero Reyno and José Á. Carsí Cubel
Software Engineering and Information Systems Research Group
Department of Information Systems and Computation
Technical University of Valencia
Camino de Vera s/n. 46022 Valencia (Spain)
E-mail: {emontero, pcarsi}@dsic.upv.es

ABSTRACT
Videogames require a more precise specification language to define and communicate gameplay than rules written in natural language. The proposed platform-independent model for videogame gameplay specification offers game designers a precise model to describe, analyze and communicate gameplay from early stages of development. The social context diagram defines how many players and teams interact with the game system. The structure diagram defines the game elements, attributes and events that compose the game system. And the rule set defines the game system behavior, implicitly specifying gameplay through precisely defined declarative rules.

KEYWORDS
Videogame Gameplay Specification, Game Design, Platform-Independent Model.

1. INTRODUCTION

Videogames currently lack a precise specification language to define and communicate gameplay [3]. Rules are the most common conceptual tool used for these game design purposes; however they are typically written in natural language, which is ambiguous and imprecise. This ambiguity creates a gap between game design and implementation since natural language game design specification is unclear and the game code implementation becomes the game specification. Re-designing games at the implementation level becomes a difficult, expensive and error-prone task, which ultimately leads to games of poor technical quality.

We propose a more precise game specification language through platform-independent models to define and communicate gameplay at a high level of technical abstraction. Redesigning games at design level is easier and conceptually cleaner than reimplementing code. In addition, the conceptual gap between game design and implementation can be easily addressed, allowing an automatic transformation from the high-level specification to other software artifacts and in turn a final compilation to code. This model-driven game development methodology reduces implementation time and errors, which ultimately leads to games of a higher technical quality.

The main goal of this paper is to provide an initial platform-independent model for gameplay specification. For simplicity, we only consider videogames, although most of the reasoning and conceptualization can be easily transferred to traditional games in general. This precise, technology-independent gameplay model is intended to replace natural language as the gameplay specification language for game designers.

2. STATE OF THE ART

Ludology, the study of games in general and videogames in particular, has pointed out the need to create models in order to explain the mechanics of games. This lack of a notation to precisely define games and game mechanics has been a traditional game design problem. Gameplay is considered the most distinguishing factor of games, since it reflects the overall experience during the interaction with the game system. Many theories and notations have unsuccessfully tried to capture the essence of gameplay in a single representation or diagram.

Frasca [5] defined video games as simulations, a software system that models the behavior of a real or fictional system. In this sense, rules define the simulation behavior. Frasca also characterized three kinds of rules: manipulation rules, goal rules and meta-rules. Manipulation rules define how players interact with the game, i.e., what players can do in the game. Goal rules define how players achieve game victory conditions i.e., how to win or lose the game. Meta-rules define how game rules can be modified, i.e., how the player can change the game. Setting aside game meta-rules, which are not mandatory for gameplay definition, manipulation rules and goal rules are a good intellectual tool for defining game system behavior. Unfortunately, this rule definition relies heavily on natural language and, consequently, lacks a clear notation and precise semantics for game behavior definition.

Salen and Zimmerman [9] also considered rules as the core element that implicitly defines gameplay. They proposed the use of three levels of rules for game design: operational rules, constitutive rules and implicit rules. Operational rules are the guidelines that players require to play the game, i.e., the rules of play. Constitutive rules are the underlying logical and mathematical structures of the game. Implicit rules are the assumed etiquette conventions of good conduct among players. Setting aside implicit rules, which again are not mandatory for gameplay definition, operational and constitutive rules are a good intellectual tool for defining game system behavior. Nevertheless, this rule definition relies heavily on natural language and, consequently, lacks a clear notation and precise semantics.

Djaouti et al [4] have identified very similar rules which appear in a great variety of games. These rules correspond to common rule templates which define gameplay. These game bricks can

be organized to build gameplay, which makes them a good intellectual tool for defining game system behavior. However they still lack a formal notation for giving a more precise meaning to each game brick or rule template.

Grünvogel [6] proposed the use of a mathematical model (Abstract State Machines) to precisely describe games as systems of objects whose state is changed by the players and other game objects. The use of a mathematical formalism to describe the game system behavior is a very precise specification method and completely removes the ambiguity of natural language. Unfortunately, game designers without a solid knowledge of this mathematical notation would have problems expressing their designs. It is important, therefore, to provide a designer-friendly representation of the game specification concepts.

Following Koster's [7] proposal of a visual, iconic and non-textual notation for game design, Bura [2] reuses a mathematical modeling language (Petri Nets) to specify rules in iconic diagrams. Nodes and links represent concrete game design concepts such as atomic transitions, texts, resource sources, storage, sinks and flows. The final diagram specification is difficult to understand and to scale, but it still gives a first approach to game design modeling without natural language.

The aim of our work is to propose a clear gameplay specification through a formal and precise rule set definition, extending all of previous research without the ambiguity of natural language.

3. A PLATFORM-INDEPENDENT GAME MODEL

A model is a simplified and abstract representation of a system that allows engineers to reason about that system focusing on its relevant details. Models are used in software development to precisely communicate key system characteristics [1]. In this case, we propose the use of models as a game design specification tool with a simplified and abstract representation of game systems.

In Model-Driven Software Engineering, a model can be independent of the specific technological platform used to implement the system. Technology implementation details are ignored at this level of abstraction and will be addressed at lower levels of abstraction either manually or through platform-specific models. In game design, platform-independent models become a technology-independent tool at a high level of abstraction, allowing clear game conceptualization without the distraction of specific implementation details.

Because many aspects of a system might be of interest, various modeling concepts and notations can be used to highlight one or more particular perspectives, or views, of that system [1]. In this multi-model approach to game design, various game design models highlight different views of interest of a game, conforming a multi-dimensional game specification. Although games have a great variety of perspectives, we propose an initial gameplay specification through the social context, internal game structure and rule set definition. Other game perspectives are outside the scope of this work and will be approached in future research. In the following subsections, we will look closely at each view of the initial platform-independent model for videogame gameplay specification.

3.1. Social Context Diagram

Videogames are a social activity which is defined in a context where players interact with each other and with a software game system. Traditionally, videogames only allowed one or two players to play the game competing with the game system. More recently, it is common to allow hundreds of online players to cooperate in a single network game. It is therefore important to precisely define how many people can play a game at the same time since the internal game system will have to compute all their inputs. The social context diagram specifies the human context that surrounds the game system with a clear and concise visual representation.

As a first social context example, let's look at the traditional competition between one or two players against the game system. Figure 1 shows the corresponding diagram. In the diagram, players are represented with a unique human node, which is complemented with a pair of sub-indexes indicating the minimum and maximum number of participants that the game allows simultaneously (one and two, respectively). The software game system is represented with a named box. It is linked to the players with a straight line, representing that the game is played by one or two players.

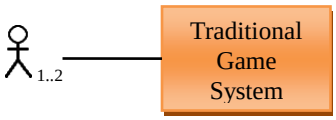

Figure 1: Example of a traditional social context diagram.

Now, let's assume that a console videogame is to be played at the same time by a minimum of one player and a maximum of four players, which is a typical game scenario in many console party games. All the players team up in cooperation against the game challenges and internally-controlled enemies. Figure 2 shows the corresponding social context diagram. The cooperation relationship encloses all the members that can play the game on the same team in a discontinuous circle. In this case, the game system box is linked to the team circle, indicating that the game is played by a team of four players instead of four individual players.

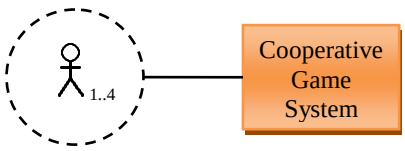

Figure 2: Example of a cooperative social context diagram.

Figure 3 shows the social context meta-model, which defines all the primitives needed to specify any game social context. Following UML notation, each meta-class in the social context meta-model specifies the structure of each social context primitive that can be used in the social context models. The player meta-class has a minimum and maximum cardinality, which defines how many individual players can access the game

simultaneously. The team meta-class also has a minimum and maximum cardinality to indicate how many teams can play the game simultaneously. Both the player and team meta-classes have a descriptive role name to portray their function in the game. The team meta-class is an aggregation of the player meta-class, indicating that a team is a group of players. The game system meta-class has a descriptive name and is associated to each player or team that can play the game. Note that the meta-model definition establishes an ontological framework to fully understand and communicate social context game concepts. From this social context perspective, players are defined as the human component of games, teams are groups of players, and the game system needs to keep track of how many players and teams can play the game simultaneously since the game system will have to cope with all their inputs.

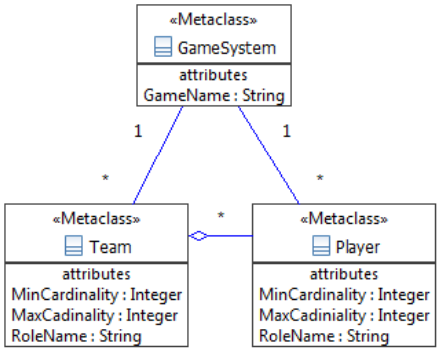

Figure 3: Social context diagram meta-model.

In order to fully understand the proposed game social context conceptualization, we present a more complex example: a soccer game. Let's assume that a multi-player soccer game is played by two teams. Each team is composed of a goal keeper and up to ten field players. Both teams compete actively to win the match. A variable number of spectators can access the game to enjoy watching the match. Figure 4 shows the corresponding social context diagram. For purposes of conciseness, when the minimum and maximum cardinalities are the same, we only write the number once, as in the team cardinality. This forces the game to have exactly two teams. Each player node is annotated with a descriptive role name which helps in the identification of player behaviors: field player, goalkeeper and spectator. The maximum number of spectators can be unlimited, which is denoted with an asterisk in its maximum cardinality.

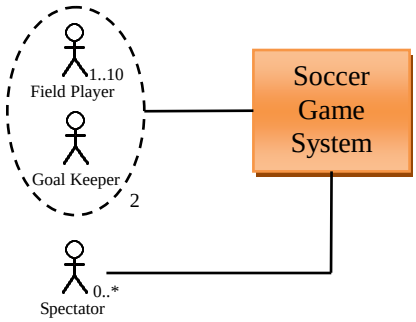

Figure 4: Example of a more complex social context diagram.

Finally, the game system is played by the two teams and the spectators. Note that the social context diagram concepts can also be applied to non-digital games, becoming a simple and powerful specification tool for traditional game design.

In order to define how each team and each player participate in the game, we need to look at the internal game system from two different game perspectives.

3.2. Structure Diagram

Games have a rigid internal structure that defines the existing elements of the game system. This structural definition is the foundation for the rules that will specify the game system behavior.

Figure 5 shows a structure diagram that defines the game elements of a simple space shooter game. Let's assume that the game (accordingly named Alien Invaders) features two spaceships, controlled by each player, that fight alien invaders in a space journey. Each player shoots and avoids space asteroids and alien invaders in order to gain points for a high score. In the structure diagram, the game system is represented with a discontinuous box that displays the game name, i.e., Alien Invaders, and includes all the game elements of the game system. Each game element is represented by a rounded box. In order to visually differentiate each game element type, all the rounded boxes have a distinctive icon. Player characters, such as the spaceships, are game characters that are controlled by a human player. These player characters are represented in the diagram with a human icon with filled-in head. Non-player characters, such as the alien invaders, are game characters that are internally controlled by the Artificial Intelligence (AI) of the game. Non-player characters are represented in the diagram with a human icon with a hollow head. The other game elements, such as the asteroids or bullets, are passive game objects that are represented in the diagram with a small black box icon. Underneath the game element icon, there is a cardinality that expresses how many game elements can exist simultaneously in the game system. The spaceship player character can be repeated once or twice, and all other game elements can exist in a variable number between zero and infinity. All game element rounded boxes have different compartments to include game attributes, events and actions.

Attributes are the descriptive data that characterize a game element. Each attribute has a descriptive name, a data type and, optionally, an initial value. The spaceship player character has integer attributes for managing the player character's lives, score and high score which are complemented with their initial values of three lives and zero points. Both the alien non-player character and the asteroid game element have an integer attribute to control the amount of points that will be added to the player character's score. Finally, both spaceship and alien bullets have no attributes.

Events define game state changes and can be invoked from the rule set to check or produce a game system change. Events have a descriptive name that implicitly defines how they behave. In Figure 5, the game system has a collide event that is supposed to check if two game elements intersect in the space. Asteroids also have a destroy event that is supposed to eliminate the game element from the system. Note that the concrete event definition is outside the scope of this design level and should be addressed

later by associating an adequate method definition to each game event. Outcomes are events a special kind that define game victory conditions such as victory, failure and draw. The spaceship player character has a failure outcome to indicate that it can lose the game.

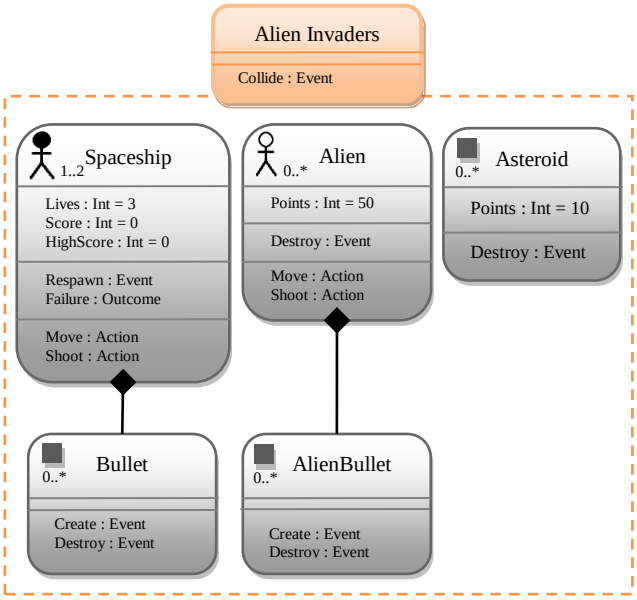

Figure 5: Example of a structure diagram for a space shooter game.

All game characters have game actions to control them, which drive the behavior of game characters. Both the spaceship player character and the alien non-player character can move and shoot, with the only difference that the spaceship is controlled by a human player and the alien is controlled by the AI of the game. Note that game events can be passive (event) or active (action). A spaceship player character can both receive a respawn event, and it can actively move and shoot. At a lower abstraction level, player character actions will be mapped to the input controllers, allowing the human players to directly control the course of the game through their character's actions.

In some cases, the game structure needs more complex primitives to express the existing relationships between game elements. It is common that some game elements own other game elements, such as the spaceship player character owning the bullets it shoots. In order to express this owning relationship between game elements in the diagram, a straight line is used to link their rounded boxes. The rhomboid end of the link points to the container game element, indicating that the spaceship owns bullets and not the other way around. Other UML relationships can be introduced in the game structure diagram, but, for the sake of simplicity, we only consider composite aggregation as a means to express the owning relationship.

Figure 6 shows the game structure meta-model that defines all the primitives needed to specify any game system structure. The structure meta-model definition establishes an ontological framework to fully understand and communicate internal game structure concepts. From this internal structure perspective, the game system contains all the game elements. Each of them has a minimum and maximum cardinality to indicate how many particular element instances can exist in the game simultaneously. The game system and each game element have a different number of game attributes and events. Game elements can also have owning relationships with other game elements. Game characters are game elements of a special type that are controlled with game actions and can be further differentiated into player characters and non-player characters, depending on whether a human player or the game AI is behind its controls.

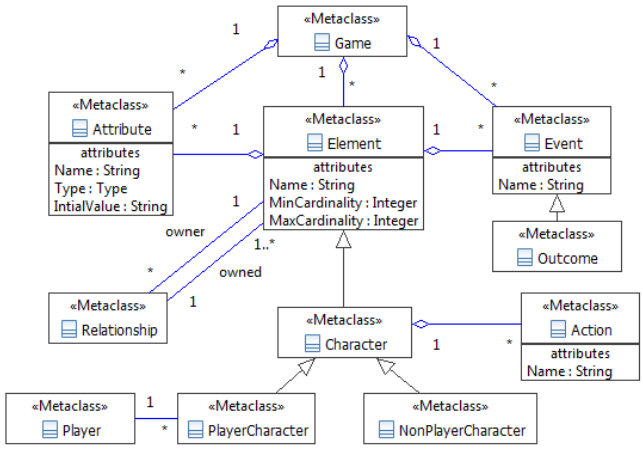

Figure 6: Structure diagram meta-model.

Note that the player character meta-class is associated to the player meta-class previously defined in the social context meta-model. This means that each player character in the structure diagram has to be associated to a human player node in the context diagram. Continuing with the Alien Invaders structure diagram example shown in

Figure 5, the spaceship player character can be related to the human node of the traditional context diagram (see Figure 1). This gives a precise mapping between player characters and their social context. Each of the two possible spaceship player characters of the internal game structure is mapped to each of the two possible human players of the social context.

With the social context and the internal game system structure precisely defined, we still need to define how the game system behaves. For this purpose, we need to define gameplay from yet another perspective.

3.3. Rule Set Diagram

Games have a complex behavior with which the players interact. The core complexity of game behavior lies inside the software game system that processes player inputs to generate game outputs. Game system behavior can be expressed with rules that implicitly define gameplay. Rules have been traditionally used as a common and widespread conceptual tool to specify game behavior. One of the main advantages of rules is that there is a higher abstraction level than with code, enabling faster and easier manipulation of gameplay. The main disadvantage is that rules are traditionally expressed in natural language and have to be manually translated to code in order to execute the game and test gameplay. Using precisely defined rules not only benefits game behavior specification and

communication, but it also allows automatic transformation to other software artifacts such as state-machines [8], bridging the gap between game design and implementation.

(1)	Spaceship.Shoot	→	Bullet.Create
(2)	Alien.Shoot	→	AlienBullet.Create
(3)	Collide (Spaceship, Alien)	→	Spaceship.Lives -= 1, Spaceship.Respawn, Alien.Destroy
(4)	Collide (Spaceship,AlienBulllet)	→	Spaceship.Lives -= 1, Spaceship.Respawn, AllienBullet.Destroy
(5)	Collide (Spaceship, Asteroid)	→	Spaceship.Lives -= 1, Spaceship.Respawn, Asteroid.Destroy
(6)	Collide (Bullet, Alien)	→	Spaceship.Score += Alien.Points, Alien.Destroy, Bullet.Destroy
(7)	Collide (Bullet,Asteroid)	→	Spaceship.Score += Asteroid.Points, Asteroid.Destroy, Bullet.Destroy
(8)	Spaceship.HighScore<Spaceship.Score	→	Spaceship.HighScore=Spaceship.Score
(9)	Spaceship.Lives = 0	→	Spaceship.Failure

Figure 7: Example of a rule set for a space shooter game.

In order to illustrate how to specify game behavior with precisely defined rules, let's continue with the space shooter game of the previous examples. Its rules can be expressed in the following natural language specification: two spaceships shoot bullets in order to destroy aliens and asteroids, gaining points for a high score. The spaceships also have to avoid aliens, alien bullets and asteroids in order not to lose their lives and consequently lose the game. Figure 7 shows the corresponding rule set with a more precise syntax and semantics. We assume that the rule set specification contains a variable number of unique rules, which are numbered accordingly in the leftmost side of the example. The rule order is irrelevant since rules can be triggered in any order. Each rule is defined by pre-conditions and post-conditions, which are the conditions that the game has to be consistent with before and after the rule is applied. In the diagram, pre-conditions are stated before the rule arrow, and post-conditions after the rule arrow. Following the first rule of Figure 7, pre-conditions state the condition that triggers the rule (such as if the spaceship shoot action occurs), and post-conditions state which effects are triggered by the rule (such as if a bullet is created). Note that the semantics of pre- and post-conditions are slightly different. A sequence of pre-conditions has to be true simultaneously for the rule to be triggered. On the other hand, post-conditions have to be true after the rule is applied, but this doesn't necessarily mean they have to occur simultaneously. Following the third rule of Figure 7, the post-conditions state that the spaceship player character has to lose one life and respawn and the alien non player character has to be destroyed. These three post-conditions have to occur after the rule is applied, but not in any particular order. The spaceship player character may react before the alien non-player character and the rule conditions still hold true. The order in which conditions are written in a rule, therefore, is irrelevant both in pre- and post-conditions.

Pre-conditions and post-conditions are constructed with all game elements, attributes, events and actions previously specified in the game structure diagram (see

Figure 5). This clarifies the rule semantics, giving a more precise meaning than natural language. The only ambiguity can be associated to the concrete semantics of game events, such as collide or destroy.

shows the rule set meta-model that defines all the primitives needed to specify any game system behavior with a rule set. Note that the meta-model definition establishes an ontological framework to fully understand and communicate internal game behavior concepts. A rule set is a collection of individual rules. Each rule is composed of one or more pre-conditions and post-conditions. Rules can be further differentiated into action rules, internal rules and goals.

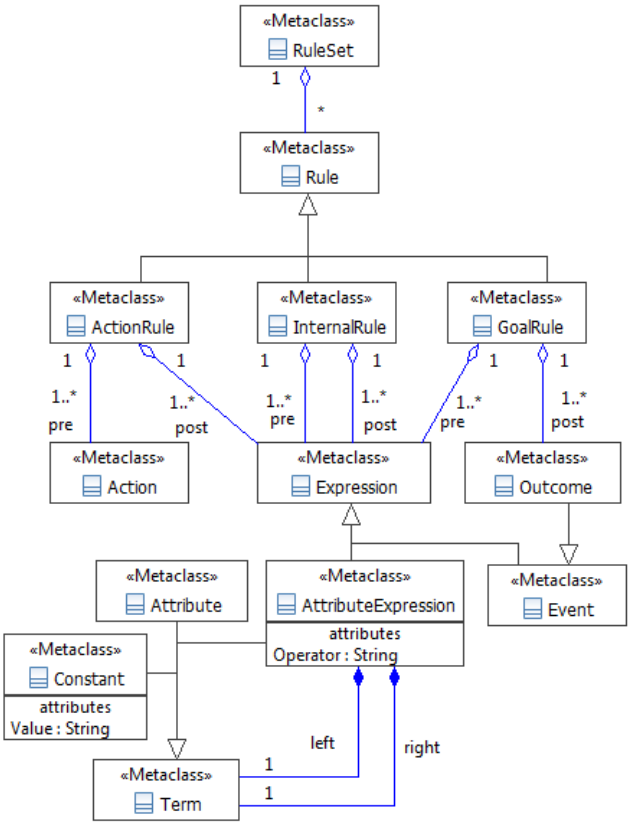

Figure 8: Rule set meta-model.

Action rules express the core gameplay mechanics that players can use through their player character actions. Intuitively, action rules express what players can do through their actions in a similar but more precise way than Frasca's [5] manipulation rules. If the spaceship player character performs its shoot action, then a bullet is created. Therefore, the player behind the spaceship can create bullets using the shoot action. The action rule meta-class defines that all action rules contain at least one action in the pre-condition.

Internal game rules express the core dynamics of the internal game system. Intuitively, internal game rules define how the game system behaves. If the spaceship player character collides with the alien non-player character, then the spaceship loses one

life and respawns, and the alien is destroyed. In internal game rules, both pre-conditions and post-conditions contain game expressions with attributes or events. An attribute expression is a statement with a left and a right side which can include logical operators, game attributes, constant values or yet another attribute expression. If the attribute expression is a pre-condition, it states a condition that must occur in order to execute the rule, such as if the spaceship player character lives are zero. On the other hand, if the attribute expression is a post-condition, it states the consequences of the rule execution, such as if the spaceship player character lives become zero. Note that, syntactically, both attribute expressions are denoted equally, but their semantics differ depending on the side of the rule in which they are placed, becoming pre-conditions or post-conditions. Game events can also be used as game expressions. If the event expression is a pre-condition, it states which event has to occur for the rule to be executed, such as if a bullet collides with the alien. If the event expression is a post-condition, it states which events occur as a consequence of the rule execution, such as if the alien is destroyed. In this case, note that some events can be used to check the internal game state (as pre-conditions) or to change the internal game state (as post-conditions).

Goals are the rules that lead to game victory conditions such as victory, failure and draw. In goals, the post-condition have to contain at least one game outcome event, and the pre-condition usually contains game expressions with attributes or events. If the spaceship player character lives are zero, then it receives the failure outcome event, consequently losing the game. Note that goals can be positive and negative depending on the game outcome that the player has to achieve or avoid in order to win the game. In the previous example, the game has a negative goal which players have to avoid. Since the game social context diagram defines that players are not united in cooperation (see Figure 1), the negative outcome event can only be applied to an individual player. This means that the first player character can lose the game while the second player character is still playing and vice versa. If the social context diagram would have specified a cooperative two-player team, the outcome could have been shared by both players using the team name in the rule definition (*Team.failure* instead of *Spaceship.failure* in Figure 7). This expressive freedom is possible because the outcome meta-class in the rule set meta-model is associated to the player character meta-class in the structure diagram meta-model, which, in turn, is associated to the player and team meta-classes of the social context meta-model. All the game perspectives of this initial platform-independent model for videogame specification are interconnected since they are intended to define the same game system behavior through different but complementary views of interest.

4. CONCLUSIONS AND FUTURE WORK

With the proposed platform-independent model for videogame gameplay specification, game designers can precisely describe, analyze and communicate gameplay from early stages of development. The social context diagram defines how many players and teams interact with the game system. The structure diagram defines the game elements, attributes and events that compose the game system. And the rule set defines the game system behavior, implicitly specifying gameplay through precisely defined declarative rules.

As this work shows, all three meta-models serve as a more precise conceptual foundation for game design than natural language. All key characteristics of the gameplay are included in the corresponding perspective, leading to a clear and concise game representation without technical implementation details. The relationships between the social context, the inner game structure and the rule set have been introduced through a simple example of game specification. Game designers now have a powerful language to express gameplay in a clear and precise specification.

Other key game perspectives still remain to be addressed in further research since the proposed meta-model only focuses on gameplay definition. These are some snapshots of the remaining game perspectives of a multi-model approach to game design specification:

- Graphic interface design defines how to visually present all game information to the player through screen layouts and game world representations. It is also important to define transitions between screens in order to specify how players navigate the game graphical interface to access information.
- Audio design defines how to convey sound and music to the player. Audio layouts allow game designers to precisely define when a sound is triggered or which music is played at each moment of the game. The set of audio design primitives includes audio events, music playlists, etc.
- Artificial Intelligence defines the behavior of all non-player characters that populate the game world, in order to express how these characters move, think, act and react.
- Game storytelling is another key perspective in story-driven games. Game designers need to specify how the story is structured and conveyed to the players using a set of narrative primitives such as story events, linear and non-linear plot structures, dialogs, scripted events, cinematics, etc.
- Level design specifies how the game world content is packaged into levels. A level editor can be automatically generated from a precise level specification, which allows a fast and easy creation of game world content.

Nonetheless, not all game perspectives can be defined independently of the underlying technology platform. Control layout design requires a platform-specific model since each technology platform (such as PCs, handheld devices, consoles and arcades) offers different controllers, ranging from the standard keyboard and mouse to highly specific controllers (such as joysticks, steering wheels, musical instruments, etc). Control game designers need to precisely specify which game actions are mapped to the specific technology controllers in order for the players to play.

All game perspectives conform to a unique game meta-model that has to be validated by professional game designers. It is also necessary to estimate the practical benefits of a precise game specification in real projects.

In summary, this multi-model approach to game design specification represents the first step in a model-driven game

development methodology where the high-level game design specification becomes the primary software artifact, replacing heavy-weighted natural language documentation in traditional game development. The platform-independent game design model will be semi-automatically translated to a more concrete platform-specific model that in turn will be compiled into code. This incremental approach keeps track of all design decisions at the different levels of technical abstraction, allowing for higher reuse and optimization of repetitive and time-consuming tasks. Model-driven game development methodology is intended to reduce implementation time and errors, which will ultimately lead to games of higher technical quality.

ACKONWLEDGEMENTS

We are grateful to the Spanish Ministry of Science and Technology since our research has been partially developed in the MOMENT project under reference TIN2006-15175-C05-01. We are also grateful to the Technical University of Valencia for granting a formation scholarship to research personnel under reference 199880998.

REFERENCES

[1] Brown, A. "An introduction to Model Driven Architecture". Available at http://www.ibm.com/developerworks/rational/library/3100.html.

[2] Bura, S. "A Game Grammar". Available at http://users.skynet.be/bura/diagrams/.

[3] Church, D. "Formal Abstract Design Tools" in *Game Developer Magazine*, August 1999.

[4] Djaouti, D., Alvarez, J., Jessel, J. P., Methe, G., Molinier, P. "A Gameplay Definition through Videogame Classification". *International Journal of Computer Games Technology*, vol. 2008, Hindawi Publishing Corporation, February 2008.

[5] Frasca, G. "Simulation versus narrative: introduction to ludology" in *The Videogame Theory Reader*, pp. 221–236, Routledge, London, UK, 2003.

[6] Grünvogel, S. M. "Formal Models and Game Design" in *Game Studies: The International Journal of Computer Game Research*, vol. 5, Issue 1, October 2005.

[7] Koster, R., "A grammar of gameplay". Available at http://www.theoryoffun.com/grammar/gdc2005.htm.

[8] Montero, E., Carsí, J. A. "Model-Driven Game Development: 2D Platform Game Prototyping". *9th International Conference on Intelligent Games and Simulation* (Game-On'08), Valencia, Spain, November 2008.

[9] Salen, K., Zimmerman, E. "Rules of Play". The MIT Press, 2004.