6
The proposed domain specific language: ontological level

This chapter defines concepts to support designing activities in pervasive mobile games, using mobile devices, sensors and actuators as the main interface elements, defining the ontological level of the proposed DSL. This corresponds to the static part of our DSL Section 6.2 presents these concepts along with their basic properties, and Section 6.3 presents the consistency rules for the concepts.

The concepts and their basic properties serve as building blocks for specifying *scenarios*. A scenario corresponds to the specification of concrete activities, including interactions among players, devices, sensors and actuators in the pervasive mobile game, being the dynamic part of our DSL. Scenarios are the subject of Chapter 7.

6.1
About the rules of the game

As discussed in Section 2.2.3, the game rules are a central concern of a game. The discussion on how to specify and apply game rules is not in the scope of this work. However, the design of activities is influenced by the rules of the game and game design. In Appendix D we provide an enhanced game design template used in the specification of our pervasive mobile game prototypes. That template provides a section to specify and detail the game rules, along with the specification of game activities.

6.2
Concepts

This section introduces some concepts that help in designing activities for pervasive mobile games. Figure 6.1 provides a summary for the concepts this section describes.

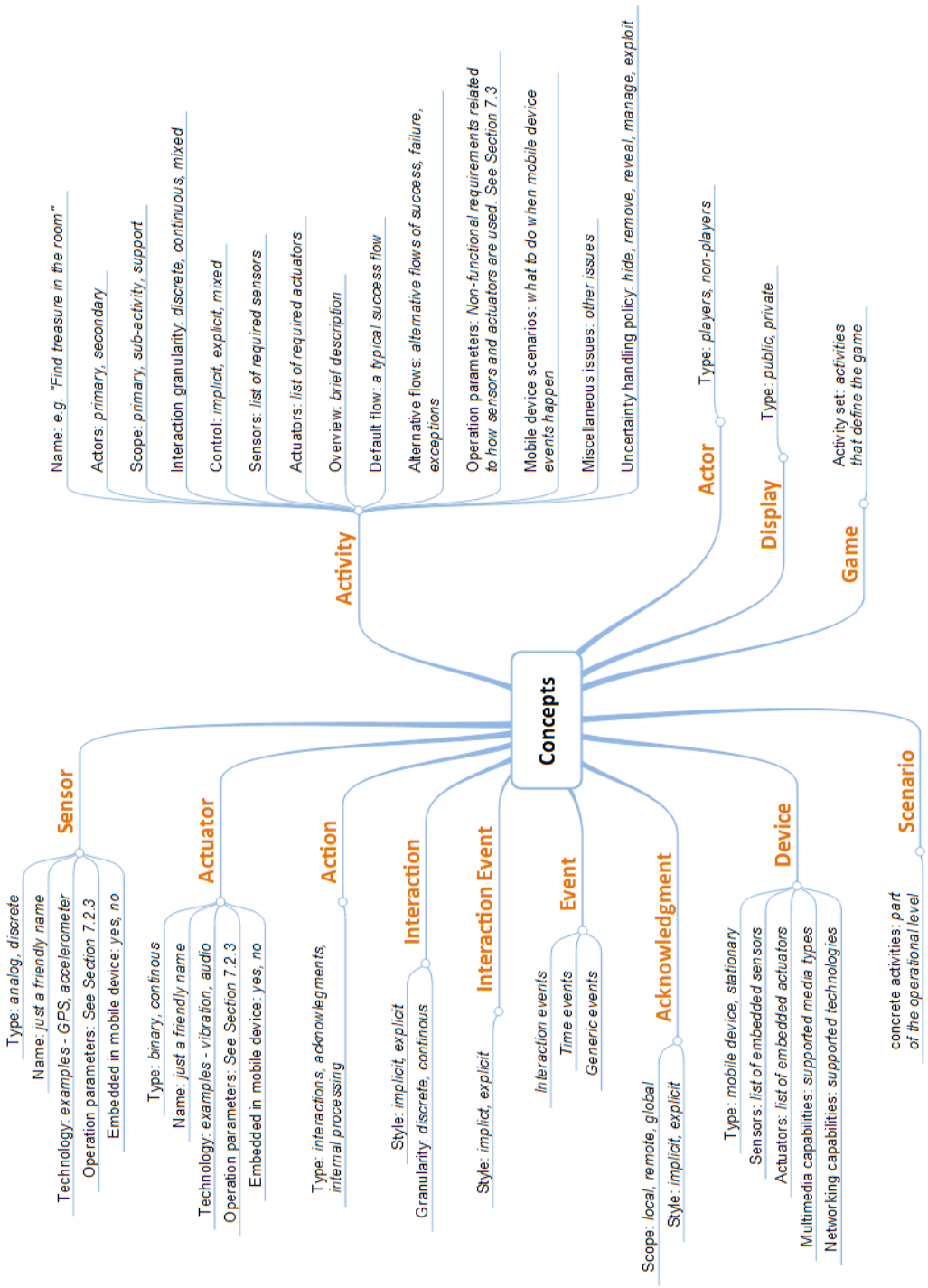

Figure 6.1: The concepts in the proposed DSL ontological level

In addition to the research on pervasive game projects, another source for the concepts was our research work on designing non-visual games for mobile

phones that use sensors and actuators as interface elements (Valente *et al.* 2008; Valente *et al.* 2009; Valente and Feijó 2011).

6.2.1 Actors

Actors are the main entities interacting with the game. This corresponds to *players* and *non-player characters*. *Non-player characters* may be represented by humans, not just virtual characters. In this case, they have *direct* participation in the game (actors representing roles, as in theater), or *indirect* participation (as sources of game content, for example). *Non-player characters* (as real people) are the subject of the pervasive game feature *Involving non-players*.

Each player controls a *device* that he uses to *interact* with the game, *sensors*, *actuators*, and other players.

6.2.2 Devices

Devices correspond to *mobile devices* or *stationary devices*. Mobile devices correspond to mobile phones, PDAs or tablets. This methodology is mainly concerned with mobile (phone) devices that have embedded sensors and actuators. Stationary devices have a fixed position in the environment. A game may deploy stationary devices in the environment to act as a container for sensors and actuators, so it is possible to design activities that occur in some local space. In case of cross-media games, players might use stationary devices to interact with some parts of the game defined by the designers. However, we are mainly concerned with situations where players control mobile devices and stationary devices are used for other purposes in the game (as providing interaction with a local space).

Devices have a set of networking capabilities – ways to establish connections to remote peers, either locally or globally. Local and global networking concepts were discussed in Section 4.2.1.

Devices also have a set of multimedia capabilities. This corresponds to the ability of producing and consuming audio, video, and other media formats.

As mobile and stationary devices are networked, they may be affected by *activities* that happen remotely.

6.2.3 Sensors and actuators

Sensors and *actuators* play an important role in our methodology, being the main interface for player-game interaction. They are embedded in mobile devices or deployed in the environment along with a stationary device (that communicates with other parts of the game).

The pervasive mobile game uses sensors to provide the context-awareness aspect of the game. The game uses actuators to send *acknowledgments* to players. Section 6.2.9 defines the concept of *acknowledgment*.

6.2.3.1 Detailing sensors

Sensors are devices that act as input sources because they respond to changes in some physical characteristic, producing a proportional output signal, which then can be measured or recorded. Table 6.1 summarizes the basic properties for sensors.

Sensor property	Arguments
Name	The sensor name
Technology	Some examples: “GPS”, “3-axis accelerometer”, “3-axis magnetometer”, “Bluetooth”
Type	“analog”, “digital”
Operation parameters	Parameters related to how sensors work
Embedded in mobile device	“yes”, “no”

Table 6.1: Basic sensor properties

Name corresponds to the name used to refer to the sensor.

Technology refers to the sensor technology. Some examples would be RFID, GPS, accelerometer, camera, ultrasonics, bar code, and magnetometer. In this case, we refer to high-level functionality of the sensor, not the hardware characteristic of the implementation.

Type refers to the characteristics of the sensor output signal, which can be *analog* or *digital.* Analog sensors produce *continuous* signals that are proportional to the measured phenomena. The signals need to be discretized through an analog to digital converter, so it possible for a computer to use them. Digital sensors produce digital (*discrete*) outputs. Therefore, the output of digital sensors consists of a finite number of different values.

Operation parameters refer to sensor operation properties to consider when specifying activities. Here is a list of them, described in detail by Bishop (2007):

- *range:* minimum and maximum range of values the sensor can output;
- *precision*: reproducibility, if the sensor measures a property a number of times, the output values should be the same in all measurements;
- *resolution*: smallest detectable change in the input value;
- *accuracy*: the maximum (error) difference between the actual value and the sensor output;
- *response time*: time required for a sensor to detect a change and produce an output;
- *sensitivity*: a ratio of how output changes when there is a change in input;
- *dead band:* input range for which the sensor does not generate outputs;
- *noise*: the amount of noise the sensor generates, usually expressed through a signal to noise ratio.

Embedded in mobile device refers if the sensor is embedded in a mobile device or not. This aspect is important because in some cases it is difficult to assess performance parameters for embedded sensors, because mobile phone specifications usually refer to sensors by their technology. In those cases, it might be necessary to perform tests to measure the performance parameters. Another issue related to this is that different mobile phone vendors might use different sensors for the same technology, and thus increasing uncertainties related to sensor usage in activities. Sensors that are not embedded in mobile phones might require a dedicated power source, which might limit the deployment of some activities.

6.2.3.2 Detailing actuators

Actuators are devices that take a command (electrical signals for example) and transform it into another physical quantity, as light, sound, force, heat, etc. Actuators act as output devices. Table 6.2 summarizes the basic properties for actuators.

Actuator property	Arguments
Name	The actuator name
Technology	Some examples: “light”, “audio”, “vibration”, “display”
Type	“binary”, “continuous”
Operation parameters	Parameters related to how actuators work
Embedded in mobile device	“yes”, “no”

Table 6.2: Basic actuator properties

Name corresponds to the friendly name used to refer to the actuator.

Technology refers to the actuator technology. Some examples would be audio, vibration, light, and display.

Type refers to the characteristics of the actuator input signal, which can be *binary* or *continuous.* Binary actuators have only two states (like on/off), where continuous actuators have a wide range (not finite) of states.

Operation parameters refer to actuator operation properties to consider when specifying activities. Here is a list of them, provided in detail by Bishop (2007):

- “Continuous power output: The maximum force/torque attainable continuously without exceeding the temperature limits.
- Range of motion: The range of linear/rotary motion.
- Resolution: The minimum increment of force/torque attainable.
- Accuracy: Linearity of the relationship between the input and output.
- Peak force/torque: The force/torque at which the actuator stalls.
- Heat dissipation: Maximum wattage of heat dissipation in continuous operation.
- Speed characteristics: Force/torque versus speed relationship.
- No load speed: Typical operating speed/velocity with no external load.
- Frequency response: The range of frequency over which the output follows the input faithfully, applicable to linear actuators.
- Power requirement: Type of power (ac or dc), number of phases, voltage level, and current capacity.”

The property *"Embedded in mobile device"* informs if the actuator is either embedded in a mobile device or not. For example, common actuators in mobile phones are vibration motors and audio speakers. As discussed for sensors, sometimes it might be difficult to have performance information for embedded actuators, meaning that it might be necessary to perform tests to assess this data.

An important consideration for actuators is that overusing it in mobile phones might drain the device memory. Also, non-embedded actuators require other power sources, which might constrain some activity designs.

6.2.3.3 A note on the operation parameters

When referring to sensors and actuators present in mobile phones, sometimes it is difficult to know in advance the operation parameters, because mobile phone manufacturers usually do not provide such detailed information. For example, mobile phone manufactures commonly list the sensors available in a device by their "technology" (accelerometer sensor, GPS, ambient light sensor, etc.). As examples of technical specifications of mobile phones for that issue, please refer to (Apple 2011; Nokia Developer 2011a; Motorola 2011).

However, one may argue that the parameters are accessible through programming APIs. Indeed, some properties of sensors and actuators may be queried or altered through the APIs, but the APIs provide a logical (abstract) view over the real hardware. Fortunately, this is the common approach, but the issue is that devices might use different components (for sensors and actuators) that may behave in different ways for the same application. For example, this is something we have experimented when testing ambient light sensors in mobile phones, where different phone models had reported different "light levels" (in the API metric) for the same input. Another related example is having the same API values (*e.g.* returning "bright" light level) for several phones when the input is slightly different.

Another important issue relating to mobile devices is energy consumption of embedded sensors and actuators, as battery is a precious resource. In general, this information is missing from mobile device specifications.

In practice, this means that it is often necessary to experiment with different devices to assess those properties, and to adapt the application to handle those issues. This is another type of uncertainty that should be handled by an uncertainty handling policy.

6.2.4 Display

Displays correspond to *private displays* or *public displays. Private displays* correspond to displays of mobile devices. *Public displays* are displays that are deployed in some environment (local space). A game might use a *public display* to promote public interaction among different players, who can be in the same local space (co-located), in different spaces, or both. In this case, the players use mobile devices to interact with the game, while the public display illustrates some aspect of the game. For example of a pervasive mobile game that uses a public display, please see *Manhattan story mashup* (Section A.14).

6.2.5 Interactions

Players, devices, sensors, actuators, displays and the game are associated through *interactions*.

6.2.5.1 Style

Player interactions have a *style* property, which can be *implicit* or *explicit.* Explicit interaction means the interaction occurs as a direct (*conscious*) command from the player, meaning that he is trying to achieve something. Implicit interaction means the interaction occurs *inadvertently* from the player point of view. Schmidt (2000) defines an implicit interaction as *"an action, performed by the user that is not primarily aimed to interact with a computerized system but which such a system understands as input"*. The *style* property is greatly related to how sensors are used in the interaction.

An example of explicit interaction would be pushing buttons on a joystick. An example of implicit interaction would be a player walking into a room, and suddenly a door opens.

A motivation for having implicit interactions is when designers want to deliver activities that might cause ambiguity, surprise, unexpectedness, disruption. It could also be used to draw attention from players to some aspect in the activity. For example, this is an approach that Roger and Muller (2006) take in their framework to design interactions to promote reflections and exploration in play.

6.2.5.2 Granularity

Interactions also have a *granularity*, which is either *discrete* or *continuous*. A discrete granularity means that the player is able to interact using *a finite set of options* for interaction. For example, pressing buttons, reading a RFID tag, positioning a device in four possible directions.

A continuous granularity means that the interaction has an *unconstrained set of options* on how to perform it. An example would be walking in a place to find WiFi access points.

6.2.6 Events

There are three types of events in our methodology: generic events, time events, and interaction events.

An *interaction event* is an important occurrence, being a *consequence* of *game logic processing* resulting from an *interaction* among actors, devices, and sensors. *Game logic* is a term commonly found in game development discussion. This term refers to the process of evaluating the game inputs and applying the game rules.

A *time event* corresponds to an event that is periodic. A *generic event* corresponds to an important occurrence in the game software, which does not fit in the other two event types.

6.2.7
Interaction events

As we are concerned with interactions, we define a sub-class of events as *interaction events*. *Interaction events* are not low-level concepts as "a reading from a sensor", or "a sensor has detected a change in some property of the environment". Instead, it is a higher level concept.

For example, suppose a game where the player uses a smartphone fitted with an accelerometer and gestures to make the virtual character walk in a virtual environment (as in our prototype *The Audio Flashlight*). The player positions the phone on a position that the game identifies as the command "walk forward". The game then tries to move the game character virtual world, but it is unable to do so because there is a virtual obstacle in the way. In this case, the game generates the (interaction) event "player hit obstacle".

An example of a generic event would be "the main game clock is over", which could mean, for example, that the players have failed to reach the game goal.

6.2.7.1
Style

Interaction and *interaction events* are closely associated. Interaction events have a *style* property that matches the style of the interaction that generates them (being either *explicit* or *implicit*).

Hence, an interaction event generated in an *implicit* interaction has *implicit style,* while an interaction event generated in an *explicit* interaction has *explicit style.*

6.2.8 Actions

An *action* is something that *affects*, *influences*, or *acknowledges* the *game state.* This can be a player *interaction*, an *acknowledgment*, or *some internal processing of the game*.

6.2.9 Acknowledgments

An *acknowledgment* (or "*ack*") is a special *action* that sends a notification to players. The game sends this notification through *actuators* or *displays.*

6.2.9.1 Scope

Acknowledgments have a *scope*, which can be *local*, *remote*, or *full. Local* acknowledgments happen in the device of the player who *has started* the activity. *Remote* acknowledgments happen in devices other than the one used to start the activity. *Full* acknowledgments happen concurrently in all devices involved in the activity, and for all actors. In this case, however, the contents of the acknowledgment may not necessarily be the same[85].

6.2.9.2 Style

Acknowledgments usually are associated with interactions. Hence, they have a *style* property that can be *implicit* or *explicit,* as the associated interaction. An *explicit* acknowledgment is associated with an *explicit interaction*, while an *implicit* acknowledgment is associated with an *implicit interaction.* The style property relates to how a player perceives the acknowledgment. Players perceive explicit-styled acknowledgments as unambiguous consequences of actions *initiated by them*, while they perceive implicit-styled acknowledgments as *unexpected* or *ambiguous occurrences*.

85 Activity specifications define the content of each *ack*. Figure 7.2 provides an example.

6.2.10 Activities

In a nutshell, an *activity* corresponds to a *flow of actions* involving *actors*, *sensors*, *actuators* and *devices* to reach some goal, along with some other properties. Activities have a well-defined *starting point* and one or more well-defined *end points*. Table 6.3 summarizes the basic properties for activities.

Activity property	Arguments
Name	The activity name
Scope	"primary", "sub-activity", "support"
Primary actors	Who starts the activity
Secondary actors	Affected/involved actors
Interaction granularity	"discrete", "continuous", "mixed"
Control	"implicit", "explicit", "mixed" (primary actor point of view)
Sensors	List of sensors used in the activity
Actuators	List of actuators used in the activity
Overview	Brief overview if desired
Uncertainty handling policy	"hide", "remove", "manage", "reveal", "exploit"
Default flow	A typical successful flow
Alternative flows	Alternative flows of success, failure, exceptions
Mobile (phone) device events	Scenarios for mobile-phone related events: "low-battery", "incoming call", "incoming message"
Operation parameters	Non-functional requirements related to how sensors and actuators are used
Miscellaneous	Other issues

Table 6.3: Basic activity properties

The remainder of this section discusses the activity properties.

6.2.10.1 Name

Name refers to the activity name that should identify it. Some examples: "Finding a treasure in the dark room", "Capture color", and "Interaction with wireless zone".

6.2.10.2 Actors

Actors (Section 6.2.1) in an activity can be *primary* or *secondary. Primary actors* are the ones that start activities. *Secondary actors* are the other actors involved in the activity. *Secondary actors* may interact directly with primary actors or not. In the latter case, *secondary actors* may be affected by consequences of actions from the *primary actor*.

6.2.10.3 Scope

Activities have a *scope*, which can be *primary*, *secondary*, or *support*. *Primary activities* have the largest scope and can be composed of *sub-activities*. Sub-activities can be secondary. A *secondary activity* can be a reusable specification for flows actions that other *primary activities* may invoke. *Support activities* have the lowest scope and are related to "house-keeping" tasks[86]. An example would be handling events related to mobile phones, as *handling an incoming call*, when the game needs to be interrupted.

Support activities can be non-interactive, and our methodology considers this variation (as an activity) to avoid providing a different concept that is similar to an existing one. This makes it possible to consider all types of activities as use cases or scenarios.

6.2.10.4 Control

From the point of view of a primary actor, *control* in activities can be *implicit*, *explicit*, or *mixed. Explicit control* means that *all interactions* are explicit. *Implicit control* means *all interactions* are implicit. *Mixed control* means there is a mixture of *implicit* and *explicit interactions* in the activity.

86 Examples of such tasks would be the ones listed in the "Mobile device events" property.

6.2.10.5
Interaction granularity

The *interaction granularity* of the activity can be *discrete*, *continuous* or *mixed. Discrete granularity* means that *all* interactions have discrete granularity. *Continuous granularity* means *all* interactions have continuous granularity. *Mixed granularity* means there is a *mixture* of discrete and continuous interactions in the activity.

6.2.10.6
Sensors and actuators

These properties correspond to a list of sensors and actuators the activity requires.

6.2.10.7
Overview

Overview is a short summary of the activity. The details of the activity are provided in *Default flow* and *Alternative flows*.

6.2.10.8
Default and alternative flows

Default flow corresponds to the typical, "happy-path" flow for the activity. Exceptions, errors, or alternative success flows take part as *Alternative flows.* An *Alternative flow* may end the whole activity, or just the flow of actions it has started.

6.2.10.9
Uncertainty handling policy

This property relates to the pervasive feature *Uncertainty handling policy* that Section 5.1.10 discusses. *Uncertainties* are inherent part of interacting with sensors and mobile networking. In this regard, activities must specify a *policy* to

handle uncertainties. This includes specifying how the designer would like to apply the policy.

This includes specifying which general strategy[87] the designer would like to use, and how he expects to use the strategy to handle the uncertainty. Chapter 8 provides concrete examples of using an uncertainty handling policy.

6.2.10.10 Operation parameters

Activities also have a set of *operation parameters* that designers must consider when using sensors and actuator. Those parameters can be considered as non-functional requirements for the activity, and are related to the operation parameters of sensors and actuators.

For example, an activity using location might require a maximum error of 10 meters in the reported location. Another example could be a requirement to "find nearby Bluetooth devices at most in 60 seconds".

Operation parameters and the *Uncertainty handling policy* are related. For example, if it is not possible (or hard) to meet a requirement of an activity (*e.g.* there is a sensor capable of providing accuracy of 10 meters, but this level of precision produces data that is too noisy), the uncertainty handling policy will have to provide a tradeoff to handle this issue.

6.2.10.11 Mobile device events

As mobile devices play an important role in our methodology, there are some important events that activities should handle.

Common to all mobile devices, there is the issue of battery life. In this sense, activities should handle low-battery warnings.

As this process highly regards mobile phones as interaction means, pervasive mobile games using mobile phones should handle high priority events related to the (communication) nature of those devices. As examples for those events:

87 From the ones discussed in Section 5.1.10.

- Incoming phone calls interrupting the game;
- Incoming text/multimedia messages interrupting the game.

In this regard, activities should be designed to be interruptible – the game application should be able to keep the game state between interruptions, not harming the player experience nor losing game data.

6.2.10.12 Miscellaneous

This corresponds to special requirements, remarks, or behaviors that do not fit elsewhere.

6.2.11 Game

For the software part of our methodology, a game corresponds to the set of activities that define it.

6.2.12 Scenario

A scenario corresponds to the concrete activity specification, being the dynamic part of the DSL. Chapter 7 discusses scenarios in detail.

6.3 Consistency rules

In this section we present a set of consistency rules for the concepts presented in Section 6.2. The rules are preceded with either *(mandatory)* or *(optional)* to denote if they are mandatory or optional rules.

1. (mandatory) activities must be context-aware;
2. (mandatory) activities must use mobile devices as primary means for interaction;
3. (optional) networked activities are optional;
4. (optional) multi-player activities are optional;

5. (mandatory) *actions* that affect the *game state* must have associated *acknowledgments*;
6. (mandatory) *events* must have associated *acknowledgments*. Hence, an event implies an acknowledgment. However, having an acknowledgment does not imply having an event;
7. (mandatory) *implicit interactions* must have associated acknowledgments;
8. (mandatory) the *style* property of *interactions*, generated *events* and associated *acknowledgments* must match;
9. (mandatory) activities must have an *uncertainty handling policy* (UHP);
10. (mandatory) activities have "operation parameters" (as non-functional requirements, "NFR") that sensors must satisfy. When this is not possible, the UHP must handle this issue;
11. (mandatory) interaction granularity in activities must be compatible with the involved sensors for the purposes of the activity.

6.3.1 Discussion

Rules 1 to 4. Characterize the domain boundary as Chapter 4 has defined.

Rules 5, 6, and 7. The rationale for these Rules is to keep the players informed about the current game state. If the game fails to inform players about the current game state, players' notion of the game state may become inconsistent. This may lead them into taking decisions that would not be supported by the game (as the game state may have changed), thus harming the player experience. Also, players may become frustrated when the game does not tell them about the results of their actions, thus degrading the playing experience.

During the research of the game *The Audio Flashlight* (Valente *et al.* 2008; Valente *et al.* 2009), we have experimented with those issues. For example, in that game the player has to tilt the mobile phone in four possible directions to walk in the virtual world. The game uses the device accelerometer to detect the gestures. Every time the game recognized the gestures, it played an audio sound effect (the

acknowledgment). As an experiment, we have diminished the volume of the audio effect so it could not be heard clearly. We have observed that players became confused about using the device, not knowing well what was going on (if they have moved or not). Later, on post-playing interviews, the players complained about this issue.

Rule 8. Requires that interactions, events and acknowledgments have compatible styles. This has to do greatly with how the game keeps players informed about the current game state, and how players perceive the results of their actions. For example, if a player interacts with a device explicitly, he would expect the game to tell him about the result of the interaction immediately (*i.e*. an unambiguous, explicit acknowledgment). However, if the player starts an interaction consciously and the game does not provide an acknowledgment, the player will become confused. This would also be the case if the game provided the acknowledgments in ambiguous ways (or as an implicit acknowledgment), while the player was expecting a direct response.

Rule 9. Requires activities to specify an uncertainty handling policy (UHP). This means the designers are aware of uncertainties, and must take conscious action to handle them.

Rule 10. Requires that the UHP must specify the behavior of the activity when it is not possible to satisfy a non-functional requirement. For example, an activity might require an accuracy that is not possible to have from available sensors. In this case, the UHP must specify how to solve this issue.

Rule 11. Relates to problems that might arise due to usability issues if this rule is broken. As an example from traditional games, requiring a player to move in an unrestricted virtual environment using the four arrow keys of a keyboard would break this rule, as the interaction has continuous granularity and the input method is discrete. However, existing research (MacLean *et al.* 2000) has discussed that using continuous input methods in interactions with discrete granular-

ity is interesting in some cases. Rogers and Muller (2006) also discuss this topic and provide as an example of interaction, “selecting a radio station using a dial”.

6.3.1.1 Variations of the rules

The objective of defining the rules is to avoid that the mentioned issues happen in the game inadvertently.

However, sometimes designers may want to create novel experiences that may twist or break some rules of the previous list. For example, Rogers and Muller (2006) have exploited uncertainties and ambiguities to create unexpected events in games for children that forced them to stop, think and reflect about what they were doing. Those cases relate to Rules 8, 9, 10, 11.

In those cases, designers should take care to not compromise player experience. The game design and setting should have means to inform players about this possibility, so players do not become frustrated when ambiguity comes in.

6.4 Summary

This chapter has defined concepts to support designing activities in pervasive mobile games, with mobile devices, sensors and actuators as the main interface elements. The concepts are: actors, devices, sensors, actuators, displays, interactions, interaction events, actions, acknowledgments, and activities. The activity concept is of central importance to the proposed DSL.

The operational level of the proposed DSL, discussed in next chapter, uses the concepts that this chapter has defined.

The chapter ends with defining and discussing a set of consistency rules for the concepts of the proposed DSL for designing activities in pervasive mobile games. A total of eleven rules have been discussed.

The consistency rules are not related to the game rules – the rules that govern the game behavior. They relate to the DSL concepts, their properties, and associations.