ATTAC-L: A Modeling Language for Educational Virtual Scenarios in the Context of Preventing Cyber Bullying

Frederik Van Broeckhoven
Dept. Computer Science - Web & Information Systems Engineering
Vrije Universiteit Brussel
Brussels, Belgium
Frederik.Van.Broeckhoven@vub.ac.be

Olga De Troyer
Dept. Computer Science - Web & Information Systems Engineering
Vrije Universiteit Brussel
Brussels, Belgium
Olga.DeTroyer@vub.ac.be

Abstract— **Cyber bullying (bullying via electronic communication tools) is a relatively recent phenomenon that especially occurs among early adolescents. As cyber bullying may have a serious impact on the mental (and physical) well-being of victims, it is important to develop effective evidence-based interventions against cyber bullying. The "Friendly ATTAC"-project has the aim to develop so-called virtual interactive scenarios (i.e. digital games) to modify behavior patterns associated with cyber bullying among youngsters. The scenarios will be developed during brainstorming sessions involving people from the different disciplines and with different backgrounds, but specialized in the domain of cyber bullying. To allow this non-technical people to specify the virtual interactive scenarios that should be developed, we have developed a domain specific modeling language that allow them to do so in an intuitive and close to natural language way. This paper presents this language in an informal way.**

Keywords—cyber bullying, domain specific modeling languages, serious games

I. INTRODUCTION

Cyber bullying (bullying via electronic communication tools [1]) is a relatively recent phenomenon that especially occurs among early adolescents. As cyber bullying may have a serious impact on the mental (and physical) well-being of victims it is important to develop effective evidence-based interventions against cyber bullying. The "Friendly ATTAC"-project [2] has the aim to develop so-called virtual interactive scenarios (i.e. digital games) to modify behavior patterns associated with cyber bullying. The target audience consists of early adolescents (10-15 year old). The games will allow youngsters, through the use of the virtual scenarios, to experience different roles (bully, victim, or bystander) in cyber bullying incidents, to react to those experiences, and to get adjusted feedback based on their individual reactions. In this way, we hope to make youngsters more aware of the consequences of certain behavior (as bully or as bystander) and help them to prevent becoming a victim. Bullies could be encouraged to cease bullying by increasing their empathy, victims could be taught adequate coping strategies and bystanders could be sensibilized to intervene. The scenarios and interactions will be based on theoretical and empirical knowledge regarding personal and contextual determinants of cyber bullying that will be obtained by performing a well-established intervention method from the field of health psychology: Intervention Mapping (IM) [3]. To realize the objectives of the Friendly ATTC project, an interdisciplinary team is established consisting of social scientists, health psychologists, computer scientists, and game designers.

One of the challenges of the project is to translate the theoretical and empirical knowledge regarding personal and contextual determinants and causes of cyber bullying into attractive virtual experiences (scenarios) for young people. The scenarios will be developed during brainstorming sessions involving people from the different disciplines involved in the project: social scientists, health psychologists, computer scientists, people working in the field, and game designers. Such an interdisciplinary collaboration requires tools and methods that can be easily used and understood by all parties involved, also non-technical users (i.e. domain-experts and possible also end-users). Such tools are not only important in the context of the project, but also later on (after the end of the project) when new scenarios need to be developed for the same or a related domain. Rather than resorting to natural language to specify the scenarios, we prefer to use a more formal approach, which has the advantage over natural language of not being ambiguous and allows to a certain extent the semi-automatically creation of the actual games. To achieve this, we will develop a domain-specific modeling language to accommodate domain experts in the design process of these scenarios/games.

A domain-specific modeling language [4] is a kind of modeling language that uses a dedicated vocabulary and provides abstractions that make the specifications of solutions easier and more accessible for domain experts. In general, domain-specific modeling languages are graphical (visual) languages because graphical or visual specifications are easier for the communication with non-technical people than textual languages; they are also helpful for conveying complex models and designs as they can help people to grasp large amounts of information more quickly than large listings of text. Such a

This research is funded by the IWT (Institute for Science and Technology) (www.iwt.be) as part of the Friendly ATTAC project

domain-specific modeling tool should not be confused with the authoring tools that exist for games and also aim to make the development process more accessible to a broader public (e.g., Thinking Worlds [5]). These tools try to avoid programming by providing a graphical and easy-to-use interface (e.g., clicking, dragging, and dropping) to compose a game. In this way, they also try to deal with the "semantic gap" between the author of the game (e.g., a domain expert or an end-user) and the game developer who each talk their own language and have their own concerns. Often, these authoring tools can generate an implementation (and in this way actual remove the need for a game implementation phase). However, in general, those tools are often limited either in the type of applications that can be specified or in the support that they provide (and often they need to be combined with scripting languages).

To try to overcome the "semantic gap" problem, some researchers ([6][7][8][9]) already proposed to consider domain-specific modeling languages (DSML) for game development. Although, DSMLs proposed for game development in general can be used as a starting point for our DSML (a virtual experience scenario can be considered as a kind of game), they are not specific enough for our purpose. A domain-specific modeling language for games against cyber-bullying should not only consider the gaming aspects but also take into consideration concepts specific to the domain of cyber-bullying, such as bully, victim, bystander, friends, as well as other objects, contexts, and behaviors specific for cyber-bullying. Identifying and defining these concepts is a first important task.

The purpose of the paper is to present our domain specific modeling language developed in the context of developing games to deal with the cyber bullying. The paper is structured as follows. The second section deals with related work. The next section discusses the requirements for the modeling language. Section 4 presents the language, and section 5 discus the limitations of the language, as well as future work.

II. RELEATED WORK

An approach close to our approach is presented in the *WEEV framework* [10]. WEEV aims towards more involvement of teachers and educators in the game development process and provides capabilities to easily create educational adventure games of the point-and-click genre. It is built upon the <e-Adventure> which is an adventure game development platform, mostly in an educational context. WEEV uses different DSVLs for the modeling of different aspects of an educational adventure game, such as the world, actors and story. Story modeling is based on an explicit representation of the interactions between the user (player) and the virtual world by means of a state-transition diagram. To reduce de overall complexity of abstractions of bigger story models, WEEV introduces extra language constructs that helps with organizing the structure. The main difference with our approach is the use of state-transition diagram. Our approach is using flow-based approach, which based on our research is easier to understand by non-programmers.

The *Storybricks*[1] framework is an interactive story design system. It features an interesting user-centered approach towards interactive story design. It provides a visual editing language based on the language "scratch" design by MIT lab[2]. Without any programming skills the users can edit the characters in the game and the artificial intelligence of the game that drives the characters. The users can setup characters and then use so-called story bricks to give them emotions, items and etc. In a similar way, Storybricks can be used to specify what need to be done at certain points in the game. In this way, an interactive scenario is modeled in an implicit way by defining a set of rules expressing which events will be evoked under what conditions. This enables the social interaction between the characters in the game without the need for it to be programmed explicitly by the designer. Our work has adopted the concept of using bricks as basic building blocks for the language from this framework.

The *80Days* project[3] aims to establish a theoretical basis for generic and engaging immersive and plausible storytelling in educational games. The project also considers cognitive and motivational/emotional aspects of learning. In the context of the 80days project, the StoryTec authoring tool [11] was improved and extended to enable creating adaptation and personalization for targeted digital educational games. Their Story Editor tool is using a visual language, which consists of story units (scene and complex scene) represented as rectangles and transition between the different connected units represented as arrows. There is a possibility to have scenes that are not connected to each other. Such scenes will be selected during the runtime depending on adaptation mechanism. The author can define the expected time that the learner will stay in a scene. Furthermore, the author could identify the skill, tasks and goals to be achieved in the scene.

III. REQUIREMENTS OF THE LANGUAGE

The DSML needs to be able to model (i.e. describe in a formal way) a virtual experience scenario. A virtual experience scenario can also be considered as a kind of educational game, as the purpose of the virtual experience scenarios is to try to change the behavior of youngsters involved in cyber bullying using principles from serious gaming. As the primary target users of the DSML are not the game designers but the (non-technical) stakeholders involved in defining the educational games, the DSML should be of a high level of abstraction. This means that it should abstract from technical and implementation details and other details that are not relevant for the purpose of defining the main ideas for the game. Therefore, based on characteristics of the target users of the DSML, we can formulate the following requirements:

1) The language should be graphical because graphical or visual specifications are easier for the communication with non-technical people than textual languages.

2) The language should be easy to learn.

3) The language should be easy to use.

[1] http://www.storybricks.com/

[2] http://scratch.mit.edu/

[3] http://www.eightydays.eu

4) The models created by the language should be intuitive and easy to understand.

As we are designing a modeling language for games, we also have the following requirements:

5) The language should be able to represent the main concepts of the game: the overall story flow; the non-player characters and objects involved in the game and their roles and characteristics; the potential actions of the player and their consequences as well as the time relationships between these actions.

As we are designing a modeling language specific for the domain of cyber bullying, we also have the following requirement:

6) Concepts from the domain of cyber bullying should be available as first class modeling primitieves.

In the next version of the language, additional requirements will be taken into consideration that deals with integrating the purpose of the games (i.e. the "educational" goal of the games related to preventing cyber-bullying) into the language

Given the requirements 2, 3, and 4 we decided to use a flow-based approach to structure the main flow of a virtual experience scenario. This decision was based on a user experiment carried out to determine which overall modeling approach was more convenient for non-technical people. However, we also decided to follow the principle of the Storybricks application that allows specifying actions and states by means of connecting bricks to form natural language like sentences.

IV. LANGUAGE DESCRIPTION

The general idea behind the language is to express the story of a game in a way as intuitive as possible. Therefore, we opted for a combination of flowcharts and natural language like syntax. The natural language like syntax will be used to specify the individual game moves in the game, while the flowchart approach will be used to express the chronological order between the game moves. By a natural language like syntax, we mean that the will use sentences that look/read like simple natural language sentences but actually have a strict syntax (to be "understandable" by the computer). Usually, our "sentences" will be composed of a subject, a verb, and a passive object. An example is "Claire talks to Tiffany". It is also possible to include an indirect object like in "Claire sends an email to Tiffany". Provided that an adequate amount of vocabulary is available, this principle can give great flexibility in defining game moves and can be understood by non-technical users in a very intuitive way. Note that, as already explained earlier, we will use a graphical representation. This means that the sentences will not be expressed as text but by means of a graphical notation.

To deal with the complexity and scalability of the models, the principle of hierarchical decomposition is used. For this purpose the concept of scenario is introduced that can be considered as an abstraction mechanism.

A. Overall Structure

The overall diagram or model representing the interactive game story is simply called *story*. Basically it consists of three parts:

- The *story flow:* intuitively this is the sequence of activities that make up the game story
- The *definitions* (optionally): intuitively this defines the game entities that are involved in the story flow. This basically comes down to describing concrete game entities or classes of game entities in terms of ATTAC-L, so they can be used in a proper way in the story flow. Note that a predefined set of definitions will be available in the language that define the entities commonly occurring in the domain of cyber bullying, e.g., a bully, a victim, a bystander, a friend.
- The *scenarios* (optionally): for a story, different scenarios can be defined (to be used in the story flow). Intuitively, a scenario defines a part of the story flow.

Furthermore, an elementary action or step in a story will be called a *game move*. Intuitively, a game move can be an action to be performed by the player or by a non-playable character, but is can also represent a state change, e.g., a mood change of a non-playable character.

B. Basic Building Blocks of the Langauge: Bricks

Regular bricks, or just plain *bricks*, are the basic building blocks for specifying *game moves*, i.e. the actual steps that will be performed in a game. The bricks are organized in different *categories* that have correspondence to certain word categories found in natural language (see Figure 1). The category of a brick is indicated by mean of a colour. In addition they can be connected to each other according to certain rules that are influenced by the grammatical rules from the natural language syntax (see Figure 2). The result is a construct that unambiguously describes a game move, expressed in a human-readable form.

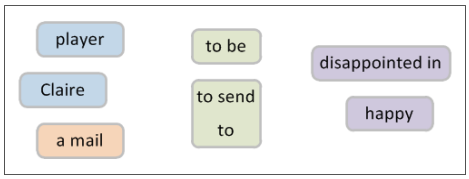

Figure 1: Bricks

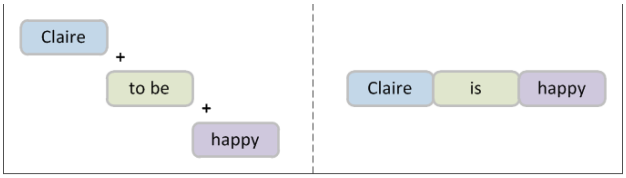

Figure 2: Composing game moves

1) Objects

Objects represent the 'tangible' game entities. They are used in a game move to refer to the game entities that are involved in the move. An object typically denotes the player, a

NPC (Non-Playable Character) or (more generally) any 'tangible' game entity.

2) Object Classes

Objects belong to classes. These classes are either predefined in the language (the classes typical used in a scenario against cyber bullying) or can be defined by the modeler in the definition part of the story. Examples of classes are person, mail, and phone. Classes can have properties. The purpose of giving the properties to classes is to be able to assign values to those properties for individual objects of that class. For instance the 'mail' class has the property 'subject'. In this way we can add a value (e.g., "Hello") to the property subject of a mail object. This is illustrated in Figure 3.

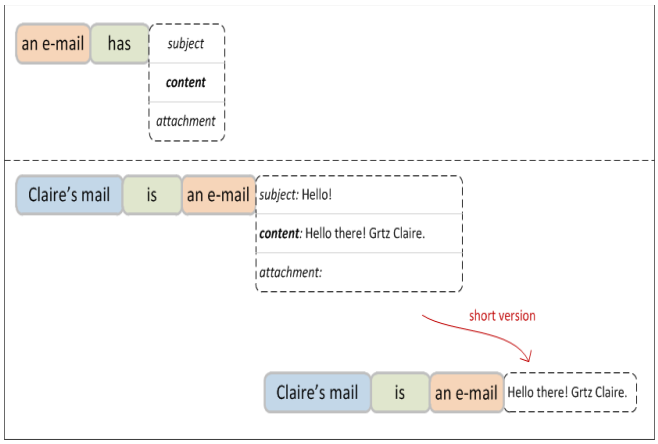

Figure 3: Example Class definition and the use of an object of this class

3) Active Vs. Passive Objects

In a game move, a distinction is made between active and passive objects. The first type, called subject in natural language, denotes the object that is responsible for the game move's action, that is, the object that performs the action specified in the game move. A passive object is also involved in the game move's action, but plays a more passive role. When the game move specifies a state change, the active object indicates the object whose state will change.

For example, when a game moves specifies that player is poking a NPC (Non Playable Character) named Claire, player is the active object or the subject of the game move and Claire is a passive object (see Figure 4).

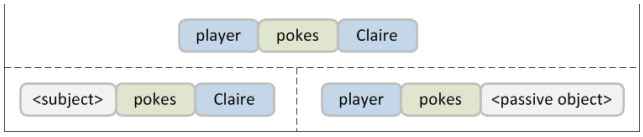

Figure 4: Example demonstrating the placement of active (subjects) and passive objects

Syntactically, subjects are always required in a game move and always mark the beginning of a game move. The presence of passive objects is dependent on the verb (see IV.B.5) that is used and is placed at the end of a game move.

4) Defined Vs. Anonymous Objects

Normally an object should be defined to use it in a game move (either it is predefined or it is defined in the definitions part of the story). This roughly corresponds to identifying an object and associating it with its meaning (class) and role in the game world. For example, when an NPC in the game world is involved in the game, an object should be defined that identifies the NPC (usually using the name of the NPC) (more on how to make object definitions in section IV.E.1). Named objects refer to concrete instances of game entities in the game world.

The player of the game is also considered as an object (note that we focus on single user games). Therefore, 'player' is a predefined object. It explicitly refers to the player of the game (which is always an integral part of the game) and can be used in the same way as any regular defined object. The availability of this object is very important as it plays a vital role in specifying interactivity.

In contrast to named objects, some objects can also be used anonymously, which means that they can be used in a game move definition without the need of their explicit definition. It is typically used for objects that are used passively in only one game move definition. These objects will typically belong to an object class coming from the target domain (here the cyber-bullying domain). An example of such an anonymous object could be 'a mail' (see Figure 5). A mail is an unnamed instance of the object class 'mail'. Often some properties (defined for the class) should be assigned to them, which can be done directly in the game move (meaning 'inline') using one or more value bricks.

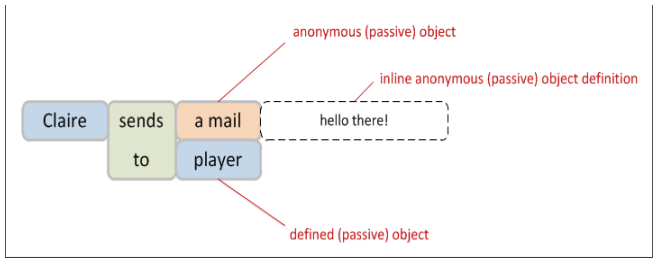

Figure 5: Anonymous vs. defined object

5) Verbs

A verb can be used for two purposes. It can be used to specify the action that a subject performs in a game move. Then, it implicitly defines what should happen at runtime, how it influences the game world. Examples are "Claire pokes Jill" or "Claire sends a mail to player" (see Figure 4 and Figure 5). A verb can also be used to indicate a state (change), e.g., "Claire is happy" (see Figure 2).

Syntactically, a verb always follows a subject. Depending on the verb used, it is optionally followed by one or more objects or adverbs (see IV.B.8).

6) Negation of Verbs

In some situations it must be possible to specify that a subject is not in a certain state. This is done by putting a not-brick between the verb and the adverb (e.g., to express "Claire is not happy") or by putting the not-brick between the subject

and the verb, e.g., to express " Claire does not know player". Both examples are illustrated in Figure 6.

7) Passive Voice of a Verb

It is also possible to use the passive voice of a verb. This essentially means that it turns the verb into an adverb (see next section). Note that the passive object of the active sentence becomes the subject of the passive sentence. This is used when the focus is on the action, and when it is not or less important or not known, who or what is performing the action. For instance, we may want to specify that "Claire is known by somebody". This can be done by using the passive voice of the verb "know". The use of the passive voice is illustrated in Figure 7.

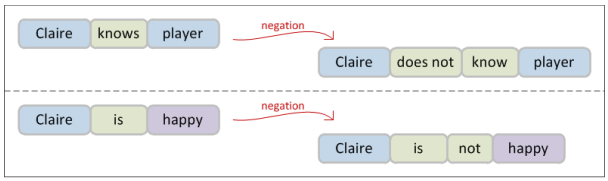

Figure 6: Negation of verbs

Figure 7: Passive voice of a verb

8) Adverb

Another category of bricks is the adverb. Adverbs are used to express states, for example to express that "Claire is happy". The adverb can also be followed by an object, as in "Claire is disappointed in player", or by a verb, as in "Claire is discouraged to bully". An adverb is always placed after a verb. Figure 8 gives some examples.

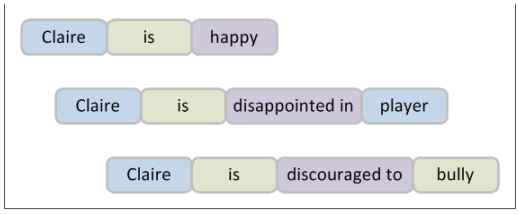

Figure 8: Examples of the use of adverbs

C. Control Structures

To express the chronological order between the game moves, control structures are used. Game moves can be sequential, or be executed in parallel, or the order can be irrelevant. Furthermore, it must be possible to express a choice and iteration. These correspond to control structures in classical programming languages.

By using the control structures we can construct so-called *storylines*. Two or more game moves that are connected by one or more control structures are called a storyline. To simplify the explanations in what follows, we will not make a distinction between a single game move and a storyline if not needed. A game move can be considered as a storyline containing a single game move. To fit with the brick principle used for the game moves, all control structures are also represented as bricks. In this paper we will explain sequence, choice, order independence, and concurrency. Because of space limitations, we omit iteration and optionality. The principle used to express them is the same as for the other control structures.

1) Sequence

A sequence is used to indicate a sequential order between game moves. It is graphically represented by a sequence brick. A sequence brick is represented as a narrow an empty white block connecting the ending and starting brick of the two consecutive game moves. When the two connected bricks are the same, the right one can be left out to get a more compact representation of the sequence. The upper part of Figure 9 illustrates a sequence between two game moves. The lower part of the figure shows how this is represented with the sequence-bricks. Note the difference between the first sequence-brick used and the second one. The first shape is the regular one. The second shape is used to connect two game moves where the beginning of the second game move (in the example "Jack") is equal to the end of the first game move (also "Jack"). In this case we can use the compact representation of the sequence of the sequence brick and the common part (in the example "Jack" is only presented once).

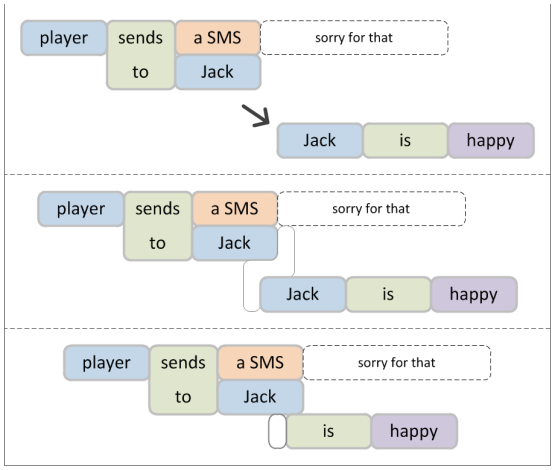

Figure 9: Composing a sequence

2) Choice

A choice defines a point in a storyline where the storyline is split into two or more alternative storylines and a decision has to be made which storyline to follow. To indicate the choice, a choice brick is used.

Syntactically, representing a choice comes down to building the different game moves around a choice brick (see Figure 10 - upper part). To avoid repetitions in the alternatives, first the common part of the alternative game moves could be identified. For example, the common part for the game moves "player pokes Claire" and "player follows Claire" is "player". However, the two game moves "player pokes Claire" and "player pokes Lizzy" have a common part "player pokes". To this common part, a choice brick is connected to which all the options are connected. The options are simply the differing parts (remaining parts) of the game moves. In the first example these are "pokes Claire" and "follows Claire", in the latter these are "Claire" and "Lizzy". See Figure 10 - lower part for

the graphical representations. If no common part exists then the complete game moves are added as options (see lower part of Figure 11). Note that an option can be followed by other game moves or in general be the start of a storyline.

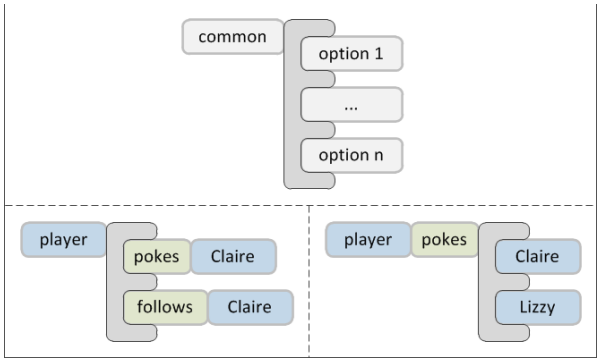

Figure 10: The use of a choice control structure

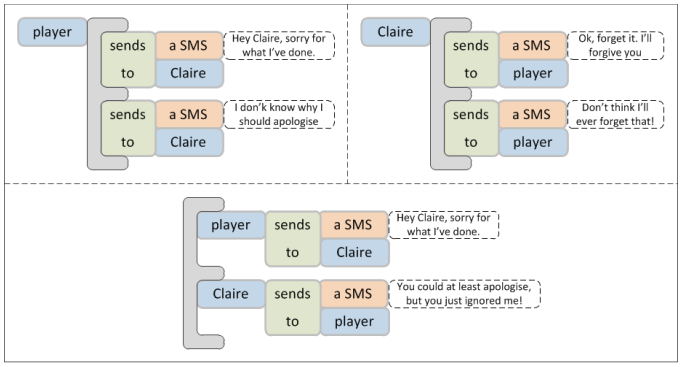

Figure 11: Examples of choice control structure

Which storyline is to be followed at runtime is dependent on the subject used in the common part. If this is the player, the decision of which alternative will be taken is completely up to the player (Figure 11 – upper left). If this is an NPC or there is no common part, the game engine will make the decision (which can be based on the current state of the game or made by the AI built into the game; Figure 11 – upper right). To accommodate the fact that it is not always possible to exactly characterize all options, an otherwise-brick is available. This allows specifying the option that needs to be followed when none of the other options could be chosen. An example is given in Figure 12.

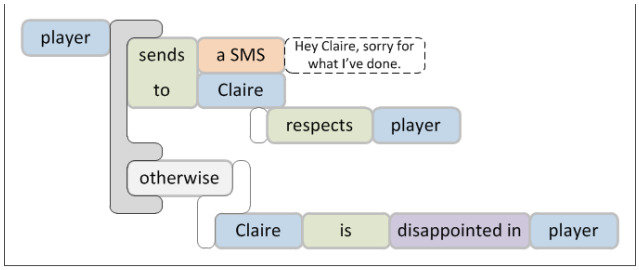

Figure 12: Example of the use of an otherwise brick

To indicate that different storylines meet again a choice join brick is used, which is a brick symmetrical to the choice brick. Syntactically, a choice join brick is used in a similar way as the choice brick, but here the different leading parts (if any) are placed at the beginning of the brick and the common trailing parts at the end. See Figure 13 for an example.

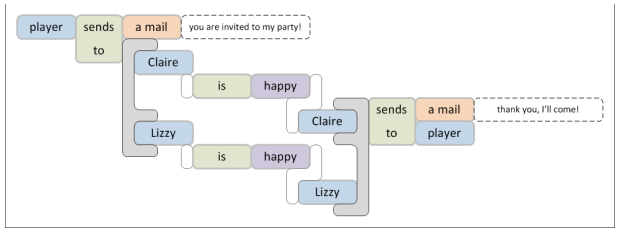

Figure 13: The use of a choice join brick

At runtime, a join has no effect on the story. It is mainly used as an organizational structure in the diagram to make the overall model of the story less overwhelming and to avoid duplications of common storylines.

3) Order Independence

An order independence structure encapsulates two or more storylines that all have to be followed, but in any order. It can be considered as a choice structure over all the possible permutations of the encapsulated storyline sections, followed by a join. The order independence brick used for the order independence is given in the upper part of Figure 14. It also shows the brick used for joining the storylines again, the order independence join brick, which is symmetrical to the order independence brick. The principles to use the brick are similar to the principles used for the choice brick. An example is given in the lower part of Figure 14. Note the use of the short hand notation for the sequence brick to indicate that the end of the first game move ("Claire") is equal to the start of the common part of the order independence structure.

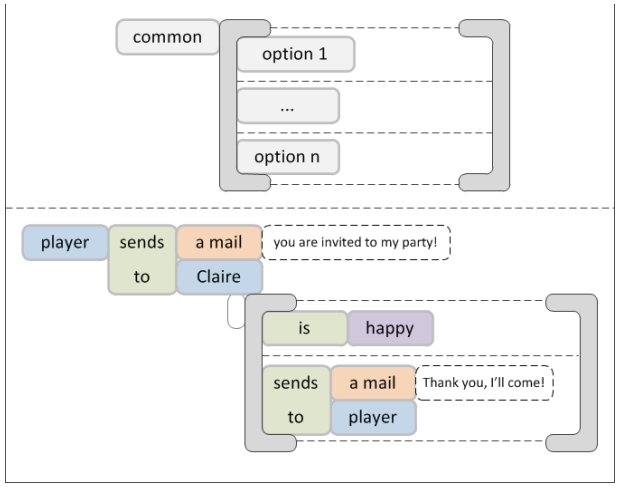

Figure 14: The use of an order independence control structure

4) Concurrency

A concurrency structure encapsulates two or more storylines that all can be followed at the same time (concurrently). This means that it is not required that one storyline must be completely finished before another one can

be started. The concurrency brick is shown in the upper part of Figure 15. The principles to use the brick are similar to the principles used for the choice brick. An example is given in the lower part of the Figure 15.

Concurrency is mostly used to allow different distinct storylines to be interwoven and to define storylines, usually initiated by non-player game entities, which seem to start 'spontaneously'. Also here the storylines can be joined again using a brick symmetrical to the concurrency brick, the concurrency join brick.

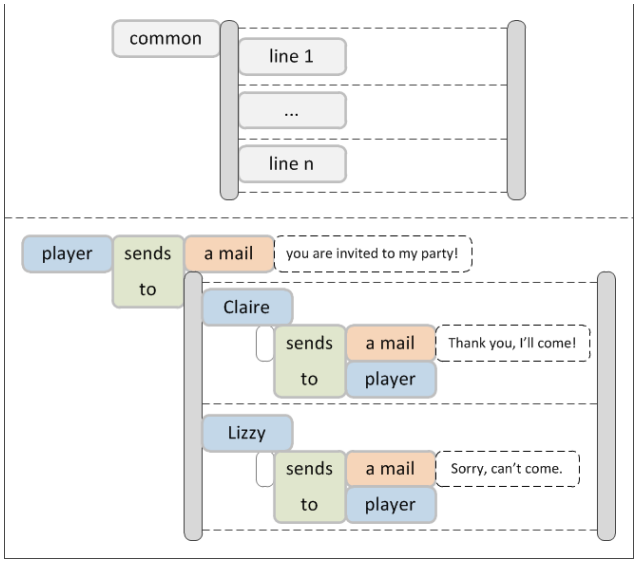

Figure 15: The use of a concurrency control structure

D. Scenarios: Hierarchical Decomposition of the Story

In order to deal with complexity and scalability, the concept of a scenario is used. It is an abstraction mechanism and allows encapsulating a part of a story. A scenario is given a name that can be used to refer to it. The name should refer to the activity encapsulated by the scenario. A special brick, called a scenario brick is used to refer to scenarios. In this way they can be used in the story flow or used to define other scenarios, i.e. a scenario brick with the name of an existing (or a still to be defined) scenario can be used to define other scenarios or can directly be used in the definition of the story flow.

Scenario bricks should be considered as placeholders for the story that they encapsulate. Figure 16 contains three scenarios: 'sending an invitation to Lizzy', 'sending an invitation to Claire', and 'sending some invitations'. The first two scenarios encapsulate some game moves. The scenario 'sending some invitations' encapsulates a concurrency between the two scenarios. This way, a hierarchy of scenarios is created that makes the diagrams smaller and makes the understanding of the diagrams easier.

A scenario is defined in the same way as a story, but it is defined within the bounds of the main story or within another scenario. This means that a scenario can also include definitions and the definitions of other scenarios.

At playtime, when a scenario is reached in the story flow, the scenario it refers to is then followed as if the scenario was directly included in the story flow. Upon reaching the end of the scenario, the story flow then continues from the scenario brick on.

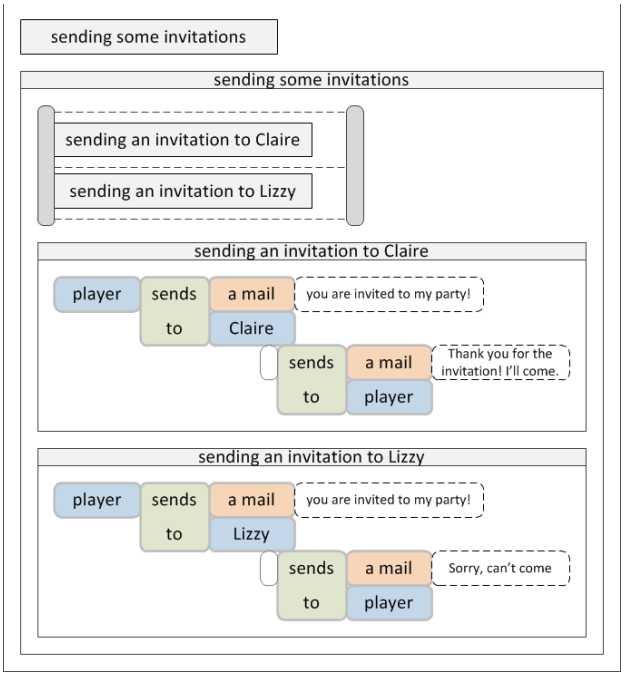

Figure 16: Hierarchical decomposition of a story by means of scenarios

E. Definitions

The definition part of a story (or a scenario) should contain the definition of all objects and classes that are not predefined in the language. In general, all predefined classes, verbs and adverbs provided for a given domain (here cyber bullying) should be sufficient to define a story for that domain.

1) Object Definition

Objects that will be used in the story should be defined, i.e. the class to which they belong should be specified and possibly also a characterization of their state. For instance, we could specify that Claire is a person, that Claire is known by the player, and that Claire is a bully. To make those specifications the choice brick and the order independence brick are used. The choice brick is used to specify alternative characteristics, e.g., Claire is friend of player or Claire knows player (see upper part of Figure 17). The order independence brick is used to sum up all characteristics, e.g., Claire knows player, and Claire is friend of player, and Claire is sad. This example is given in the lower part of Figure 17.

2) Object Alias Definition

The definition part can also be used to define an alias for an object or for a group of objects. For instance, we can define 'stranger' as a person that does not know the player or who is not known by the player. This example is given in Figure 18.

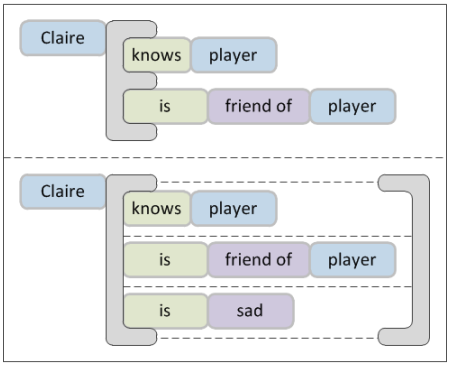

Figure 17: The use of choice and order independence bricks to indicate alternative and all-inclusive characteristics

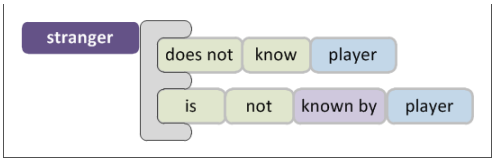

Figure 18: An object alias definition

F. *Story Flow*

The story flow part of the story specifies the actual sequence of game moves that make up the game story. In practice, this will be done by linking scenarios using the different control structures provided in the language. This will give a high level view of the story flow: the main structure and decomposition of the story flow into scenarios. To obtain more details, the different scenarios need to be inspected. It would of course also be possible to directly specify the complete story flow in term of game moves and control structures. However, in general, this will result in a very large diagram that is difficult to understand and to maintain (see Figure 19).

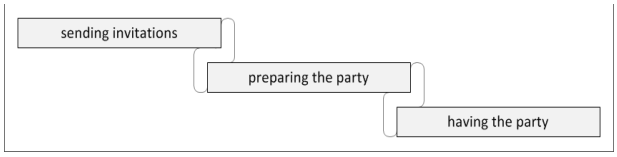

Figure 19: A story flow example (in terms of scenario bricks)

V. Limitations and Furture Work

The current version of the ATTAC-L language has the capability to express story-based games. However, it also has still some limitations.

- There is not yet a way to refer in the game story to events and actions that took place during the game play, e.g., we cannot base a decision on past actions. For instance, it could be useful to state what should happen if a certain message has been send. We will investigate if the past tense of the verbs can be used for this.
- All objects are defined at design time. This means that it is not possible for the player to define his own objects at playtime, for instance his own friends. It is not clear if this functionality will be required for the kind of virtual experience scenarios that will be developed in the context of Friendly ATTAC. It is also not yet obvious how to add the functionality to the language without complicating it considerably.

Our future work will focus on applying the language in the context of the Friendly ATTAC project to actually define the virtual scenarios that will be developed in the context of the project. This will provide feedback on the expressiveness and intuitiveness of the language and will allow us to improve the language. To use the language in practice, tool support is necessary. First we will develop a graphical tool that allow to easily compose the scenarios using ATTAC-L and that enforce the syntax of the language. Next, we will investigate code generation for the language to speed up the implementation process of the actual scenarios.

References

[1] Q. Li, “A cross-cultural comparison of adolescents’ experience related to cyberbullying”, *Educational Research*, vol. 50, nr. 3, pp. 223–234, 2008

[2] “Friendly Attac”. Available: http://www.friendlyattac.be/en/.

[3] L. K. Bartholomew, G. S. Parcel, en G. Kok, “Intervention mapping: a process for developing theory- and evidence-based health education programs”, Health Educ Behav, vol. 25, nr. 5, pp. 545–563, okt. 1998.

[4] J. Luoma, S. Kelly, en J.-P. Tolvanen, “Defining Domain-Specific Modeling Languages: Collected Experiences”, in Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling (DSM 04), 2004.

[5] Thinking Worlds | Rapid Sims & Games Creation. Available: *http://www.thinkingworlds.com*

[6] J. Dobbe, “A Domain-Specific Language for Computer Games”, Delft University of Technology, 2007.

[7] A. W. B. Furtado en A. L. M. Santos, “Using domain-specific modeling towards computer games development industrialization”, in 6th OOPSLA Workshop on Domain-Specific Modeling (DSM’06), 2006, p. 1.

[8] R. Guerreiro, A. Rosa, V. Sousa, V. Amaral, en N. Correia, “UbiLang?: Towards a Domain Specific Modeling Language for Specification of Ubiquitous Games”, pp. 449–460, 2010.

[9] E. J. Marchiori, J. Torrente, Á. Del Blanco, P. Moreno-Ger, en B. Fernández-Manjón, “A Visual Domain Specific Language for the Creation of Educational Video Games”, IEEE Learning Technology Newsletter, vol. Vol. 12, nr. No. 1, pp. 36–39, 2010.

[10] E. J. Marchiori, Á. del Blanco, J. Torrente, I. Martinez-Ortiz, en B. Fernández-Manjón, “A visual language for the creation of narrative educational games”, Journal of Visual Languages & Computing, vol. 22, nr. 6, pp. 443–452, dec. 2011.

[11] S. Gobel, L. Salvatore, en R. Konrad, “StoryTec: A Digital Storytelling Platform for the Authoring and Experiencing of Interactive and Non-Linear Stories”, in International Conference on Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS ’08, 2008, pp. 103 –110