ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/267801341

Towards a Domain Specific Modelling Language for Serious Game Design

Article

CITATIONS
14

READS
487

6 authors, including:

Stephen Tang
Liverpool John Moores University
30 PUBLICATIONS **688** CITATIONS
SEE PROFILE

Martin B. Hanneghan
Liverpool John Moores University
44 PUBLICATIONS **802** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Combining the Internet of Things and Serious Games (Smart Serious Games) View project

Project Utilising the Third Dimension to Visualise Cyber Security Requirements in BPMN View project

All content following this page was uploaded by Martin B. Hanneghan on 25 March 2015.

The user has requested enhancement of the downloaded file.

Towards a Domain Specific Modelling Language for Serious Game Design

Stephen Tang
Kolej Tunku Abdul Rahman
Jalan Genting Kelang, Setapak,
53300, Kuala Lumpur, Malaysia.
+6016 293 5691
mr.stephentang@gmail.com

Dr. Martin Hanneghan
Liverpool John Moores University
James Parson Building,
Byrom Street, Liverpool, L3 3AF, United Kingdom.
+44 151 231 2577
m.b.hanneghan@ljmu.ac.uk

ABSTRACT

Computer games are fun to play but the design and engineering tasks involved behind the scenes can be daunting to many. Model-Driven Engineering offers an approach to simplify the technical complexity of computer games development and tools that support model-driven development can help to reduce the barriers towards the adoption of games-based learning. This paper describes the process to define a Domain Specific Modelling Language (DSML) for serious game design modelling. Existing software modelling languages are analysed to determine suitability for use in serious games modelling. A modelling framework for serious games design is then proposed taking account into requirements of serious game design and findings from the analysis.

Categories and Subject Descriptors

I.6 [**Simulation and Modelling**]: General

K.3.1 [**Computer and Education**]: Uses of Computer in Education

General Terms

Documentation, Design, Languages.

Keywords

Serious Game Design, Game Modelling, Framework, UML, MDE, MDD.

1. INTRODUCTION

Computer gaming has become major part in the development of many children, teenagers and young adults. It offers more than mere entertainment to game-players. In the past computer gaming was regarded as an unproductive activity that promotes aggressive behaviour [1], smoking [2], obesity [3] and poorer academic performance [4]. However many are beginning to realise the potential of computer games for non-entertainment purposes.

The application of gaming to education and training is an attractive approach to persuade and motivate learners to acquire knowledge in a specific domain. Computer games of this kind are generally termed *serious games* and are used as content for *games-based learning*, a technology-assisted learning approach which uses computer games as platform for learning and teaching. Many are captivated by this approach and yearn to adopt this to realign pedagogy to fit the new generation of digital-savvy learners. Findings from initial studies has shown that computer games can be used to acquire certain cognitive abilities and improve their understanding in topics presented [5, 6]. These preliminary results are convincing and have gained tremendous interest from different sectors including government, academia and industry to further explore the benefits of such opportunities [6, 7]. Many also agree that it is now appropriate to take advantage of gaming technologies to create a new generation of educational technology tools to equip learners of all ages with necessary skills through experiential learning [7].

However the adoption of games-based learning is affected by the source of appropriate computer games with pedagogy elements i.e. serious games [8]. Development of games-based learning content may require a huge budget and such financial barriers have been a major challenge for many teachers who intend to adopt it. Some have tried sourcing such content from commercial-off-the-shelf (or COTS) games and this has proved very challenging to get the right content for use in an educational setting.

Although the application of computer games in the area of education and training may differ from other domains, engineering serious games software remains unchanged. The underlying technologies of serious games are the same as mainstream entertainment games. Distinctions are notable in the definition of rules, play and aesthetic representation (usually defined by game designers but in the context of serious games this is the role of teachers). Game developers rely on game technology platforms, game engines and other custom tools to produce games quickly, affordably and reliably. However using these tools present a very steep learning curve for non-developers and may require a substantial amount of knowledge of the technology behind the game which can be a huge challenge to most teachers who have little knowledge about game development. A user-friendly high level modelling environment that provides maximum software assistance to teachers allowing them to modify properties of instances that suit their lesson design could lower the barrier of adopting games-based learning.

This paper presents the roadmap undertaken in this research study to define a high level modelling language for serious game design. Section 2 highlights a brief discussion of model driven engineering for serious games. A conceptual framework for serious games design is proposed in Section 3 which serves as a basis for the analysis of major system modelling techniques in Section 4. The system modelling techniques are assessed against their suitability for use in

ISBN: 978-1-902560-21-2 © 2008 GDTW

modelling serious games with a focus on education and training. Section 5 presents the framework proposal for serious games modelling and conclusions are drawn in Section 6.

2. MODEL DRIVEN ENGINEERING

Model Driven Engineering (MDE) techniques use Domain Specific Modelling Languages (DSMLs) to formally represent structure, behaviour and requirements of a particular domain. Aspects of models are then analysed and translated through transformation engines and generators to synthesize software artefacts consistent with the models [9]. The past few years has witnessed the software industry encouraging the use of models to architect the structure of complex software systems [10] based on the concept of software industrial production. Research on MDE mainly focuses on the mainstream general software industry – business enterprise solutions, as efforts to reduce complexity in development using modern platforms.

2.1 Serious Games and MDE

At present the game industry uses object-oriented (OO) game design and scenario-oriented level design approaches in the process of writing computer games [11]. Some early efforts to promote the use of models in designing games includes the use of UML use-case diagram and class diagrams to represent games [12] and even modifying the UML use-case diagram to cater for game flow design purposes [13]. These OO game design approaches are relevant and easily translatable to game software design. Although a majority in the software industry unanimously agrees to the use of OO Modelling as the benchmark [14, 15], recent findings suggest that the UML 2.0 standard is rather complex and practically limiting the application of it into Model Driven Development (MDD) environments [16]. Therefore there is a need to investigate the use of other system modelling techniques available (or even define new techniques) for use in MDD environments for serious games development. It is also important to note that the majority of teachers from various domains who wish to produce serious games have little (or no) technical skills or knowledge in software development.

2.2 Benefits of MDE

Some of the notable benefits of MDE from a software development context include an increase in productivity, promotion of interoperability and portability among different technology platforms and support for generation of documents and software maintenance [17]. In the context of serious games development, MDE techniques can help to formally gather the necessary requirements of a serious game through formal modelling notations and later be translated into source codes specific to a game technology platform. Such an approach encapsulates the technical aspects of serious games development and therefore provides the necessary aid to teachers to produce their serious games with minimal low-level technical knowledge in game development. A Model-Driven Development environment for serious games would require a game modelling language, a framework that can integrate arts assets and technical features of serious games such as physics and artificial intelligence (AI) and transformation engines. Such a development environment could help lower the barriers to games-based learning while also helping to produce serious games of an appropriate quality.

2.3 Defining the DMSL Process

The MDE process involved in defining a DSML for serious games design adopts an evolutionary approach proposed by Tolvanen [18] which are categorised into four phases;

1. Identifying abstractions and relationships;
2. Specifying the language concepts and associated rules (meta-model);
3. Creating the visual representation of the language (notation);
4. Defining the generators for model checking, code, documentation, etc.

The remaining section of this paper reports our progress made so far (mostly in phase 1, phase 2 and early stage of phase 3 of the above process) in an attempt to apply MDE techniques to serious games development. A conceptual framework proposed in Section 3 extends the studies done in [19, 20] that describe the abstractions and relationships of serious games. The conceptual framework will serve as a basis for development of a DSML for use in a serious game MDD environment.

3. A CONCEPTUAL FRAMEWORK FOR SERIOUS GAME DESIGN

Ideally serious games can have learning material embedded within narrative and storytelling which facilitates learning through game-playing or role-playing. By studying the properties and behaviour of in-game components and the relationship between these in-game components in the scenario defined encourages problem solving abilities [21].

Art assets and technical functionality such as physics and artificial intelligence that represent the behaviours of avatars and objects in the game world should ideally be abstracted from the design requirement. This de-emphasis should permit teachers to model the game based on available assets and features developed professionally to maintain the quality of the application. Teachers should be able to customise certain properties and features of objects available instead of struggling to create new assets that fit into the game world. However we acknowledge that there are still certain aspects of serious games that require teachers to exercise higher levels of control and expertise such as the placement of avatars, objects and event triggers although we expect a fair proportion of others to be assisted through the use of visual templates.

3.1 Serious Game Design Architecture

Our conceptual framework consists of three tier design architecture; *flow*, *scenarios* and *objects*. Each tier focuses on certain aspects of pedagogy in serious games based on Gagne's 'Conditions of Learning theory'.

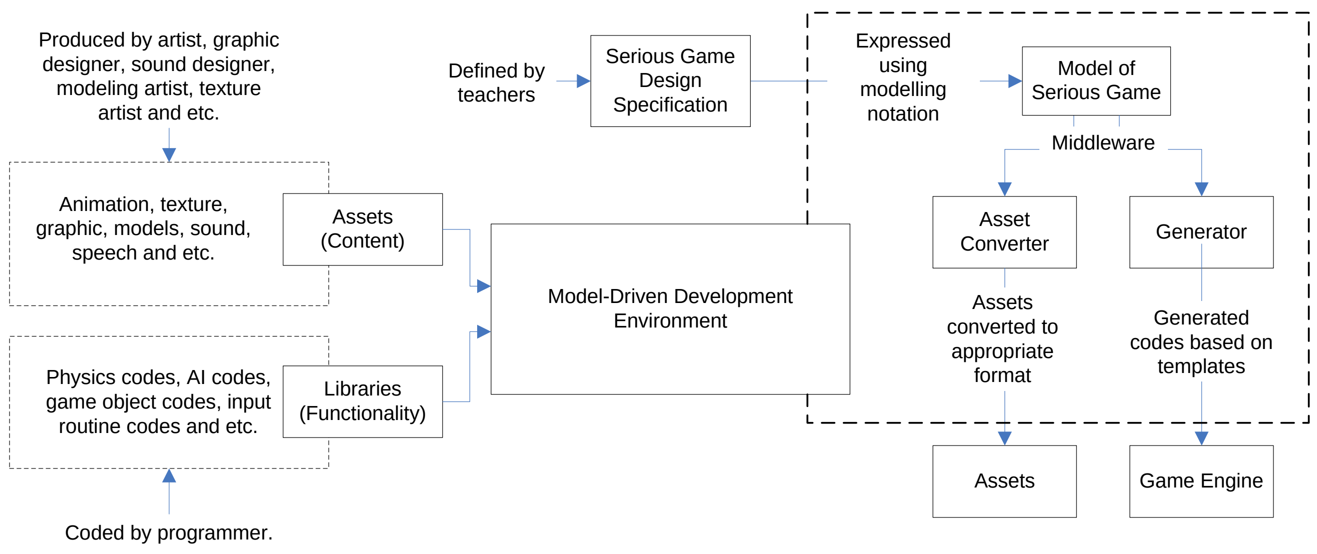
Produced by artist, graphic designer, sound designer, modeling artist, texture artist and etc.
Animation, texture, graphic, models, sound, speech and etc.
Assets (Content)
Physics codes, AI codes, game object codes, input routine codes and etc.
Libraries (Functionality)
Coded by programmer.
Defined by teachers
Serious Game Design Specification
Model-Driven Development Environment
Expressed using modelling notation
Model of Serious Game
Middleware
Asset Converter
Generator
Assets converted to appropriate format
Generated codes based on templates
Assets
Game Engine

Figure 1. Software Architecture for Model-Driven Serious Game Development.

3.1.1 Flow Design
Flow design mimics the lesson plan where teachers plan the order of the activities of learning for a lesson or a period of study. This design task would include careful planning of events that take place prior to the actual learning activity in serious games such as *gaining learners attention*, *informing learners of objectives* and *stimulating recall of prior learning*. Game flow can be constructed by linking various types of game screens. Transition can either be triggered through user interaction or a timer event.

3.1.2 Scenario Design
Scenario design represents the construction of a situation which consists of characters, objects, objectives, scripted events and problems to be solved through game-playing. This design task involves design of instructional events such as *selective perception*, *providing learning guidance*, *eliciting performance*, *providing feedback*, *assessing performance* and *enhancing retention and transfer*. Problems in a game scenario can be designed with simple learning tasks such as identification of objects and interactions with objects to learn about properties and behaviour of objects to complex tasks such as the specific ordering of interactions and operation of complex or aggregated virtual objects to solve problems (e.g. operating a virtual 'machine' object. Alternatively a game scenario could be fully scripted and be used as an animation component based on the idea of machinima production as described in [21].

3.1.3 Object Design
Object design involves customisation of characters and objects that are used in designing a specific game scenario. Customisations are limited to lists of vital information, actions and reactions, and intelligence (if required). Teachers can choose from the available model *(mesh)*, animation, physics and AI functionalities to represent the identity of the character or object.

3.2 Components of Serious Games
Serious game components can be represented in four broad categories, namely *screen*, *graphical user interface* (GUI), *in-game object* and *scenario*. These components are organised in a meaningful way on *screens* and are linked to provide *flow* in the serious game.

3.2.1 Screen Components
Screens are a virtual canvas used for rendering graphical elements with functional or non-functional behaviour to represent different sections of a serious game. It can be used for construction of menus, to display information to learners (such as instructions, synopsis of the level, objectives, results or even alternative learning resources), configuration of inputs and simulation of a scenario. Screens can represent a viewing perspective of an environment occupied with avatars and objects and directed event flow that are associated with objectives. GUI components may appear on the game screen to update learners on vital information about the game. They may also provide linkage to other screens that display inventory of items and additional information to the scenario (i.e. hyperlinked information).

3.2.2 Graphical User Interface (GUI) Components
GUI components are visual controls that allow the user to execute specific functions or present information graphically or visually in a serious game. GUI components with behaviours can respond to mouse or keyboard events (such as mouse-click, mouse-release, mouse-rollover, mouse-rollout, key-press and key-release). Events signify the type of input captured from learners and are linked to a response that is often presented visually or aurally. Decisive events such as mouse-release and key-release can trigger screen-related functions or in-game-related functions depending on the mode of usage. Common GUI components for use in screen mode include buttons, list boxes, radio buttons, scroll-bars and textboxes, whereas in scenario mode, most GUI components are made of custom graphic buttons. Other static (or non-interactive) GUI components include text and image which are used to display

information such as instructions, objective statements and in-game statistics.

3.2.3 In-Game Components

In-game components refer to visual representations that are in a scenario. It can be classified into four distinct classes; *character, vehicle, object* and *sensor-based object*. Each class may have variations in type due to the differences in behaviours and properties.

3.2.3.1 Characters

Characters are social-beings within the game-world that could be represented in various forms such as humans and animals. Although serious games are meant to reflect some form of reality, representing characters in other forms such as an electron and microorganism can provide different perspectives for learners. A character consists of *vital state* that represents the value of existence in the game world such as health, energy and money; *action* which represents the range of actions (including reflexes) in motion and speech (which can be represented in sound or text format); and *avatar* or visual presentation of the character. Teachers should be able to define the necessary vital states and select the necessary motion, speech and avatar from a list of assets to represent the character. For Non-Player Characters (NPCs) teachers can select predefined behaviours that react to learners' interaction.

3.2.3.2 Vehicles

Vehicles transport characters and objects from a source location to destination location. Typical forms of vehicle include two wheels, n-wheels or as extended carrier.

3.2.3.3 Objects

Objects represent items that could be used as *tools* or *enhancements* to a character, or *consumables* that would improve the vital states of a character in certain aspect. Tools or enhancements can be visually attached to avatar whenever learners choose to use it and are useful in helping learners to solve problems in a given scenario. Consumables however disappear from the game-world when interacted with. Consumables may also be associated with certain tools or enhancements such as *money* may be associated with a *bank* object.

Objects can also be composite objects. This provides the ability to create machines from objects that are composed of sub-components each of which has their own state and behaviour. This facility provides a means for emergent behaviours in objects within a persistent game world over time.

3.2.3.4 Sensor-based Objects

Sensor-based objects have mechanistic features that can interact with the environment. Example of sensor based objects includes automatic doors and land mines.

3.2.4 Scenario Components

Scenario components consist of building blocks for environment, situation and evaluation of learner's actions. The design of environments and situations is regarded as defining learning objectives although the tasks are comprehensive, whereas evaluation of learners is related to the assessment according to the defined learning objectives.

3.2.4.1 Environments

Environment related components are separated into groups of *indoor* and *outdoor* components. Indoor components include structural components such as floors, walls, ceiling, stairs, doors and windows, and decorative components such as sofa, table, vase and carpets. Outdoor components are terrain and obstacles such as lamppost and postal box to trees, bushes and buildings. Most of these environment related components are static and those that support interaction may be regarded as sensor-based objects of in-game components.

3.2.4.2 Situations

Situation related components consist of *in-game components, event triggers* and *event acts*. In-game components (as described in Section 3.2.3) can be placed within the *environment* and constructed to set up a situation before adding in event triggers. There are two distinct types of event triggers; *time-based triggers* and *collision-based triggers* which will execute an event act which is a list of actions composed by the teacher to express a situation. Another component required in modelling situations is the *location marker* which is placed within the environment and used as a destination when moving in-game components in an event act. The teacher can define the list of *event acts* based on the available actions of an in-game component placed in the scenario.

3.2.4.3 Evaluation

Evaluation of a learner's actions is monitored by an *observer*. Each interaction are reported to the *observer* and evaluated based on the evaluation list defined by teacher. A list of interactions can be grouped into a *goal* or *sub-goal* and be evaluated individually or as a whole. Interactions can also be measured in ordered or non-ordered form. Values captured can then be displayed in screen as a form of feedback to learners on their performance.

3.2.5 Pedagogic Structure for Serious Games

At present four pedagogic structures have been identified for used in serious games namely *complete game structure, presentation-based structure, training-based structure* and *scenario-based structure* as described in [19] with improvement made those the structures.

3.2.5.1 Complete game structure

The *complete game structure* embeds all lessons into a single software application that is distributable as courseware for self-learning purposes. The structure consist of lessons designed accordance to Gagne's nine instructional events (described in 'Condition of Learning Theory') using the components of serious games described in previous sections. Lessons can be composed of interactive and non-interactive elements such as static screens, animated presentations, guided tutorials and scenarios that require problem solving. Teachers will have to decide the best combination to bring about the optimal learning experience to learners.

3.2.5.2 Presentation-based structure

A *presentation-based structure* is a subset of complete game structure which uses game technology to create animated scenarios for use in teaching and learning. In such structures interactivity is kept to minimum. This approach may have one or more presentations logically grouped.

3.2.5.3 Training-based structure

A *training-based structure* guides learners to use or operate certain objects such as tools and machinery that in reality might be considered dangerous or costly for training purposes. It can also be used to provide training on the standard procedures of a task. It is more useful as a form of instruction rather than providing a means for assessment of learning.

3.2.5.4 Scenario-based structure

A *scenario-based structure* is based upon the representation of a single scenario reconstructed to reflect reality involving characters and objects in a virtual space. It is a stripped down version of the complete game structure that consists of a single scenario only. Learners are expected to apply the knowledge and skills they have learnt either from serious games or from traditional classroom lessons in the scenario to solve prescribed problems. It possesses similar characteristics to the training-based structure except that learners are not guided in performing tasks and solving problems. This structure is ideal for learners to experiment their solutions and for domain experts to measure and access a learner’s understanding toward the area of study.

4. SOFTWARE MODELLING LANGUAGES FOR SERIOUS GAME DESIGN MODELLING

Modelling languages allow practitioners to visualise, specify, construct and document domain specific artefacts. Software engineers use modelling languages to model software systems including business logic and system functionality. Models are then translated by programmers into software code. Various modelling languages, both textual and graphical, have been developed for use in modelling software. A broad selection of these includes Z-notation, Petri-nets, IDEF, Business Process Modelling Notation (BPMN) and Unified Modelling Language (UML) to name but a few. Most of these software modelling languages are useful to game software development but not necessarily useful in modelling game design. The aforementioned software modelling languages are analysed against criteria defined in Section 4.1 to shortlist those which is suitable for modelling serious game design.

4.1 Criteria

Based on the proposed conceptual framework in Section 2 a modelling language should provide the necessary semantic notation to model aspects of serious games such as *objects*, *events*, *relationships*, *transformation of states*, *flow*, *interaction* and *temporal* requirements. The criteria also include some characteristics of formal methods proposed by Clarke, Wing, et. al [22] which are deemed useful in this study:

- The modelling language requires little (or no) modification for use in modelling serious game design.
- The modelling language is able to represent all the necessary game design requirements.
- The modelling language provides instant benefits, such as increase in productivity, to its practitioner.
- The modelling language is translatable to a programming language.
- The modelling language is easy to learn.
- The modelling language is easy to use.
- The modelling language provides orientation towards error detection.
- The modelling language supports evolutionary development to extend its functionality and support.

4.2 Analysis of Software Modelling Language

Z-notation is based on Zermelo-Fränkel set theory and two-value predicate logic for describing and modelling computing systems [23]. Z-notation is powerful and can be used to model various aspects of serious games. Models are represented using mathematical notations and are translatable to programming languages easily. It also permits orientation towards error detection but may require strong background in mathematics and therefore has a steep learning curve for new-adopters.

Petri-nets are directed bipartite graphs used particularly in model checking, graphically-oriented simulation, and software verification [24]. Petri-nets may be suitable for modelling artificial intelligence in serious games but lack support for object definition and temporal aspects in modelling events for serious game design. In addition Petri-nets are also mathematically demanding to practitioners.

BPMN[1] is used mainly in modelling business process. It is represented with a small set of graphical notations which are group into flow objects, connecting objects, swimlanes and artefacts that can be used to model flow and business process. BPMN can be a suitable candidate for modelling serious games due to simplicity of the diagramming approach while remaining clear and concise in representation of requirements. The only issue however lies in the support for defining attributes for in-game components (properties and functional behaviour).

IDEF[2] (Integrated Computer-Aided Manufacturing Definition Language) is a collection of modelling methods for function, information, data, process decision, OO design and ontology description. In the context of modelling serious games IDEF1 (used for data modelling) and IDEF4 (which focuses on OO design) are suitable for modelling in-game component. Other diagramming methods in the IDEF family are less suitable for use in modelling serious games. Although IDEF is a well-developed modelling language it can be perceived to require a steep learning curve compared to UML [25].

UML is the industry standard general software diagramming language which consists of thirteen types of diagram to document three aspects of software systems; functional, structural and behavioural. It is one of the ideal candidates for use in modelling serious games. The OO modelling approach makes it simple for translation to OO programming languages. Class diagrams can be used to model in-game components and relationships between the components in a scenario as described in [12]. An extended version of use

[1] Read more about BPMN from www.bpmn.org

[2] Read more about IDEF from www.idef.com

case diagrams has been used for modelling scenarios [13]. Scenarios can also be modelled using interaction diagrams. Transformation of states for in-games object such as artificial intelligence and animation can be modelled using state diagrams.

Among the system modelling languages reviewed, UML stands out as the best candidate for modelling serious games. Z-notation and Petri-nets are relatively robust modelling languages but both are mathematically demanding for users (in this context referring to teachers with no technical knowledge of computer games development) and therefore less favourable. BPMN provides a simplistic modelling approach which is suitable for teachers but however lacks essential aspects of object modelling. IDEF on the other hand has limited support for time-constraint definition. UML is suitable for modelling in-game components and relationships between the components in a scenario. Although UML meets most of the criteria defined in Section 4.2, it still requires slight modification before it can be used for modelling serious games. UML by definition and its sheer range of diagramming conventions is also complex and may not be suitable for use in other domains such as serious games design [26]. Using UML in game development however provides great benefits where game developers can have a unified view, but may be rather overwhelming for teachers who may not have experience in modelling. This suggests there is a need for a DSML for serious games instead of adopting an existing generic method.

Other approaches towards formal game design also include the use of Soft Systems Methodology [8, 27] and the MDA (Mechanics, Dynamics and Aesthetics) framework [28]. These approaches are more suitable for modelling game design; however the approach is too generic as there are no notations or rules that define the modelling process but rather guidelines that helps designers think creatively. Without formal rules in modelling it is difficult for models to be translated into a programming language. However these are valid formal approaches that can aid game designers to think systemically about the game.

5. A MODELLING FRAMEWORK PROPOSAL FOR SERIOUS GAME DESIGN

Findings from analysis of software modelling languages suggest that there is a need of a high level modelling language which is able to encapsulate the complexity and technicality of game software in addition to the criteria defined in Section 4.1. Our proposed modelling framework uses UML class diagrams and an extended version of the state diagram as a basis for modelling flow and in-game components. In general modelling of serious games components can be categorised into *data modelling* and *visual modelling*. *Data modelling* mostly involves definition of objects, flows and processes whereas *visual modelling* involves positioning of in-game components in the virtual world, construction of an environment and layout of GUI components on-screen. In some aspects of serious games modelling visual modelling is crucial to enable teachers to provide necessary information to be used in data processing. Although a portion of serious games design will involve organisation of visual elements this framework will focus mainly on visual modelling that provides positional data of in-game components that is useful in serious games design.

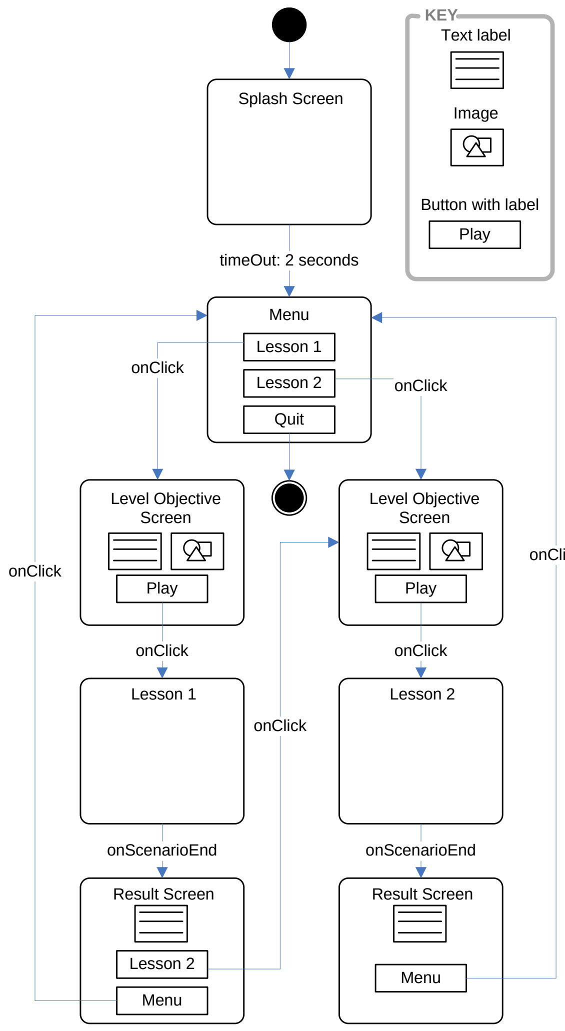

Figure 2. Extended version of State Diagram for flow design modelling.

5.1 Modelling flow for screen components

Screen component models should be able to represent the flow of interaction as both an overview and using the finished look as feel aspects of the application. This can be represented using diagrams similar to state diagrams in UML that have been extended to include GUI modelling aspects as illustrated in Figure 2. Screens can be designed with GUI components as described in Section 3.2.2 in which some are linked to content (text or image) and where interaction events (key press or mouse click) can serve as the transition condition in the state diagram. Transition conditions can also be automatically triggered for example a countdown timer or end of game event that initiates the transition. Visual representations of GUI components are included in the state diagram and transitions are labelled with appropriate UI events such as *onClick* (mouse-click event), *timeOut* (end of timer count down) and *onScenarioEnd* (end of game scenario). Connections to screen objects are supported through UI components such as buttons depending on the type of UI event used to trigger the transition. Detailed screens can then be modelled in a visualisation environment and use content holders such as text labels and image areas assigned with an image source.

5.2 Modelling in-game components

While UML class diagrams are suitable for modelling in-game components, behavioural aspects still require low level modelling for example capturing AI behaviour. The AI model proposed by Kienzle et al. [29] uses a variant of Rhapsody state-charts along with UML class diagrams and sequence diagram for formalism. This approach is useful for experienced AI modellers but can be rather complex and demanding for non-technical users. The approach requires one to model the states of various AI-related components such as sensor, analyser, memoriser, strategic decider, tactical decider, executor, coordinator and actuator. These components in the model are loosely coupled and hence components with low level of abstraction such as sensors and actuators can be replaced with a more specialised variant while high level abstraction components, e.g. the strategic decider, can be reused in most cases.

An alternative to the approach proposed by Kienzle et al. and used in this framework is the Viable System Model (VSM). The VSM can be used as a model for representing in-game components as a whole rather than just from a behavioural perspective. Early attempts to apply VSM to game design [30] has provided insights to enable us to map the VSM for representation of in-game components as illustrated in Figure 3. This representation is modelled on Cannon-Bard Theory which suggests that humans feel emotion first then react upon them [31]. Although VSM is said to be bureaucratic and its detractors say it may not be suitable for application to computer games, we believe its application in this context serves as a useful model for architecting in-game components. The VSM in UML class representation is described as follows:

- System V (*Personality Manager*) represents the method that initializes the in-game component based on some defined personality.
- System IV (*Sensor Manager*) represents methods related to collision detection which are triggered at appropriate intervals.
- System III (*Action Generator*) can be considered as a set of pre-defined methods for updating vital data, action states and AI-related components such as path finding and agent related behaviour that are enabled (or included) in the definition of personality traits of in-game components.
- System II (*Acting Manager*) represents a method that receives messages from System III and distributes these messages to System I (*Speech Generator*, *Motion Generator* and *Appearance Generator*) for action to be taken.
- System I components represent the main action entities. These entities carry out the basic mechanics of the game. The *Speech Generator* in System I represents a method that plays an appropriate sound file. *Motion Generator* represents the physics and animation methods related to the in-game component modelled. *Appearance Generator* then represents the method that renders the graphics to the screen.

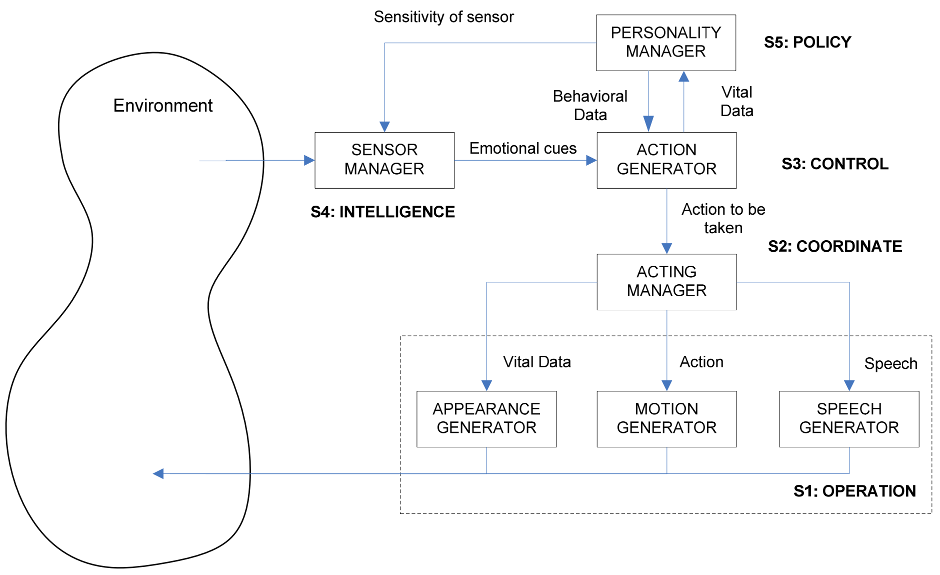

Figure 3. VSM for In-Games Component.

The VSM can be used as a high level abstract model to represent in-game components and by its recursive nature can either expose or hide the lower level functionalities are defined by developers. Instead of modelling states for various AI components mostly from scratch [29] teachers can alter the rules of predefined states to model the behaviour of in-game components described in the VSM as System III (Control). This can be done through definition of personality which in turn is used by System V (Policy) to govern an entity as a whole. Definition of personality involves specification of the variables, assets and data sources, relationships with other in-game components (as in which objects are related) and reaction when vitals are at critical state or when related objects are sensed. Reaction sends a message to the *observer* which monitors each interaction. Using the VSM as a model allows in-game components to be more easily modelled encapsulating technical knowledge from domain modellers.

When modelling in-game objects such as tools used for concentrated force (hammer, axe and wooden stick) System IV will only be enabled when the object is actually used by a character. Sensor-based objects and vehicles in contrast do require System IV definition to operate properly. Relationships between other in-game components are represented as part of the personality that acts as a filter to the *Action Generator*. User controlled character *Action Generator* entities can be replaced with *Action Controllers* which takes user input and trigger the appropriate action.

5.3 Modelling Flow for Game Scenarios

Modelling game scenarios requires teachers to construct an environment and place the necessary in-game components into this virtual space. This task requires a visualization environment that allows the teacher to set up the scenario before game flow can be modelled (i.e. placing the necessary location markers and event triggers). One approach towards modelling game flow is the use of use case diagrams by extending it with the use of decision tree [13]. This is a useful approach to model game flow and interaction from the player's perspective but it does not provide support for defining actions for other in-game components during an event. Support for definition of actions for in-game components in a scenario is important for serious games as it is used for executing specific actions to represent a scenario similar to the role of a director giving instructions to the actors. AI for Non Player Characters (NPCs) is still important to represent a spontaneous reaction to the player action during game-play. Also there is no support for teachers to monitor the types of interactions which will be used to measure the learner's progress against the defined learning objectives.

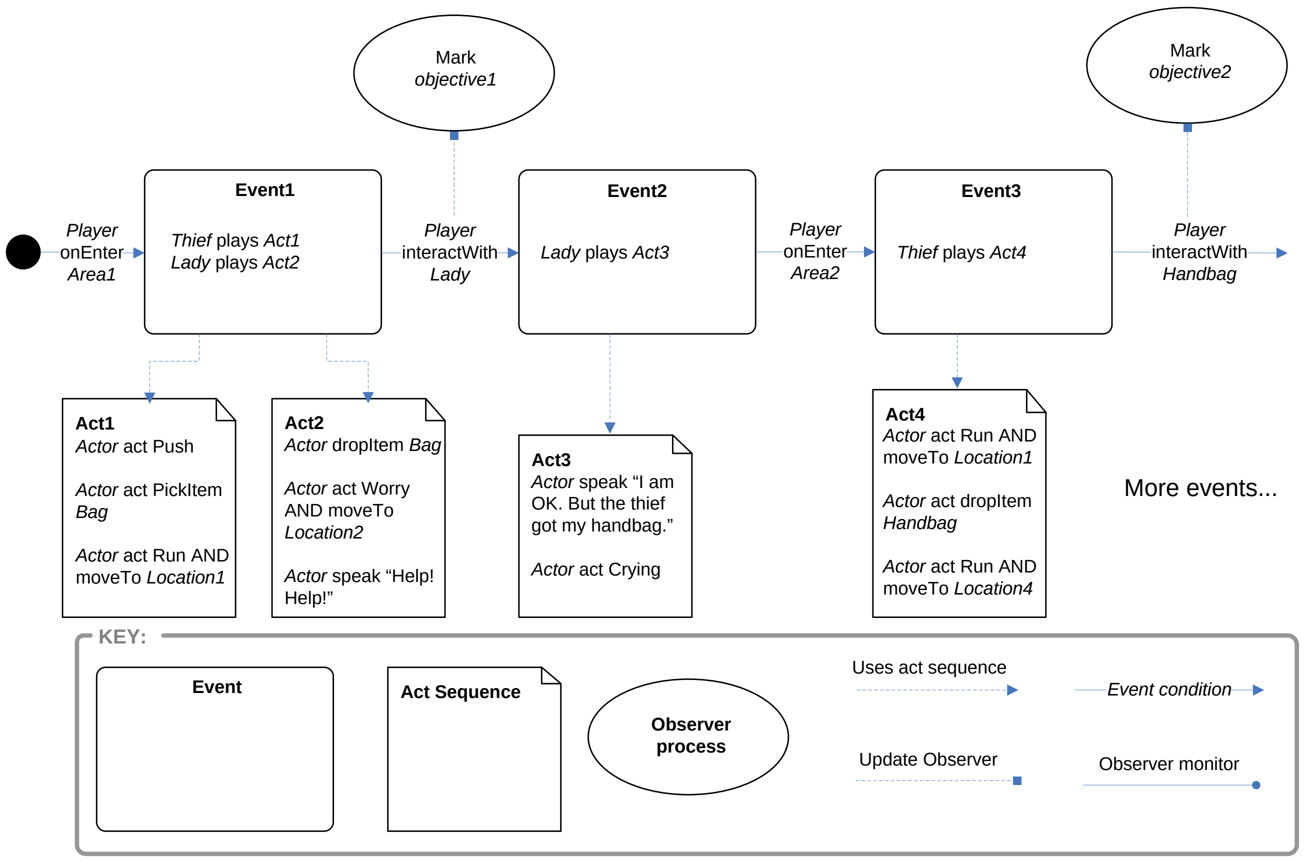

Figure 4. Extended State Diagram that models partial flow for "Thief robbing a Lady" Scenario.

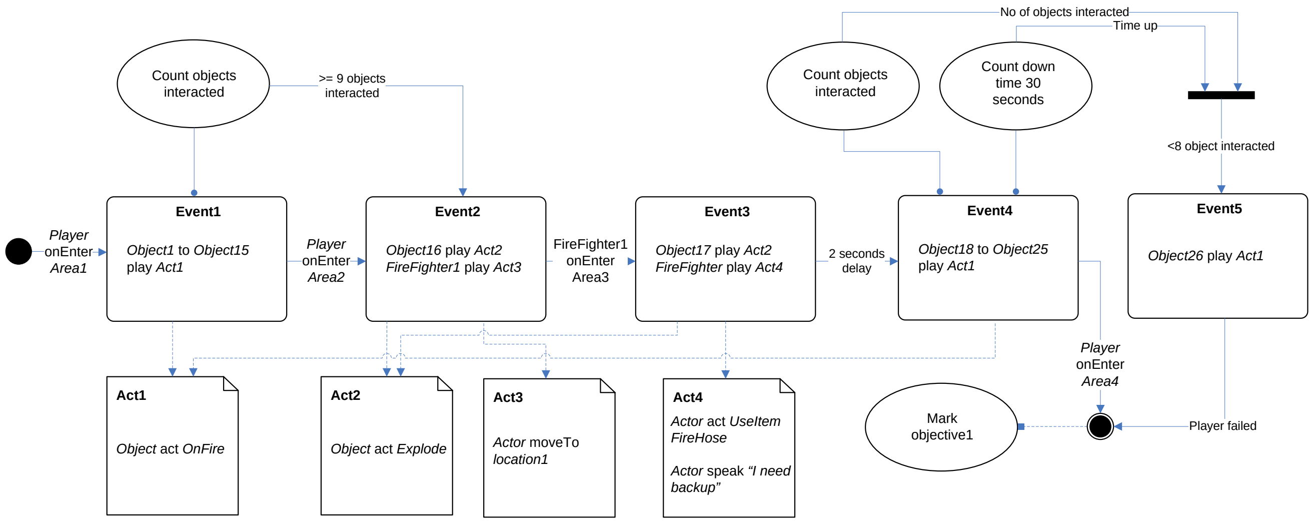

Figure 5. Extended State Diagram that models flow for "Factory Fire breakout" Scenario.

The approach proposed in this framework uses a UML state diagram with additional notation to support definition of act sequences for in-games components and communication with observer components. Rounded boxes represent event states, rectangles with a folded corner represent act sequences used in an event by an in-game component while ovals represent an observer process involved. A dashed-arrow line denotes which act sequence is used in an event, a dashed-line ended with square bullet represents an update of state by the observer, a solid line with an arrow represents an event condition while a solid line ended with round bullet indicates a continuous monitoring process of an observer. Flow within a diagram is indicated with a solid line and should be read according the direction of the arrow. Each diagram represents one scenario and can be merged with (or drilled down into) the flow design diagram in Figure 2 to provide a complete view of the game. Each event has a list of in-game objects that are directed to act within the virtual scenario based on the acting script assigned as illustrated in Figure 4. Events take place in the virtual world when a player enters a particular area and acting begins. Events happen once unlike the approach used in [13] which may happen more than once when players repeatedly visit the same are in the virtual world. Events can be immediate when players enter certain segments or introduced with delay or make use of an observer to introduce a secondary event when a certain variable observed reaches a pre-defined value or range. As an example, a serious game which involves fire-fighters that are trying to save a factory on fire, if nine fire-spots or more (assuming there are fifteen fire-spots defined in strategic location in the virtual world) have been put out introduce a new event that act as explosion from the control room (see Figure 5). Every interaction sends a message to the observer, a component which monitors and records every interaction, for immediate action to be taken or delayed until later.

6. CONCLUSION

Model-driven serious games development is a promising approach to reduce the complexity of games development and lower the barriers toward games-based learning. The framework proposed in this paper presents on-going research in this field which aims to define a high-level abstraction to include a wide variety of serious game applications. The intention is also to simplify the modelling process by excluding most of the low level activities of software modelling to provide a clear separation of assets and code from design. At this stage, the proposed modelling framework may appear still too technical to non-technical persons but will be made more user-friendly via the modelling environment as we focus on representing more of the technical aspects in GUI notations rather than software code. Furthering this research study a middleware descriptive language will be defined based on the proposed modelling framework for use in serious games authoring environments. Our authoring environment currently under development will be tested and results from testing can then be used to improve the modelling framework and the modelling environment.

7. REFERENCES

[1] C. A. Anderson and B. J. Bushman, "Effects of Violent Video Games On Aggressive Behavior, Aggressive Cognition, Aggressive Affect, Physiological Arousal, And Prosocial Behavior: A Meta-Analytic Review of the Scientific Literature," in Psychological Science, vol. 12, 2001, pp. 353 - 359.

[2] D. Kasper, S. Welsh, and C. Chambliss, "Educating Students about the Risks of Excessive Videogame Usage.," ERIC, 1999.

[3] K. Subrahmanyam, R. E. Kraut, P. M. Greenfield, and E. F. Gross, "The Impact of Home Computer Use on Children's Activities and Development," The Future of Children, vol. 10, pp. 123-143, 2000.

[4] M. R. Hauge and D. A. Gentile, "Video Game Addiction Among Adolescents: Associations with Academic Performance and Aggression," presented at Society for Research in Child Development Conference, Tampa, Finland, 2003.

[5] M. d. Aguilera and A. Mendiz, "Video games and education: (education in the face of a "parallel school")," in Computers in Entertainment (CIE), vol. 1, 2003, pp. 10-10.

[6] BECTa, "Computer Games in Education: Findings Report," vol. 2008, 2006.

[7] FAS, "Harnessing the power of video games for learning, Summit on Educational Games 2006," 2006.

[8] A. E. Rhalibi, M. Hanneghan, S. Tang, and D. England, "Extending Soft Models to Game Design: Flow, Challenges and Conflicts," presented at DiGRA 2005 - the Digital Games Research Association's 2nd International Conference, Simon Fraser University, Burnaby, BC, Canada, 2005.

[9] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," Computer, vol. 39, pp. 25-21, 2006.

[10] A. Brown, "Part I: MDA and today's systems," in An introduction to Model Driven Architecture, vol. 2008, 2004, (Online) http://www.ibm.com/developerworks/rational/library/3100.html (Accessed 10 September 2008)

[11] V. Gal, C. L. Prado, S. Natkin, and L. Vega., "Writing for video games," presented at VRIC 2002, Laval France, 2002.

[12] C. S. Ang and G. S. V. R. K. Rao, "Designing Interactivity in Computer Games a UML Approach," International Journal of Intelligent Games and Simulation, vol. 3, pp. 62-69, 2004.

[13] M. J. Taylor, D. Gresty, and M. Baskett, "Computer game-flow design," Computers in Entertainment (CIE), vol. 4, pp. Article No. 5, 2006.

[14] G. Miller, S. Ambler, S. Cook, S. Mellor, K. Frank, and J. Kern, "Panel - Model driven architecture: the realities, a year later," presented at Conference on Object Oriented Programming Systems Languages and Applications, Vancouver, BC, CANADA, 2004.

[15] G. Miller, A. Evans, I. Jacobson, H. Jondell, A. Kennedy, S. Mellor, and D. Thomas, "Panel – Model driven architecture: how far have we come, how far can we go?," presented at Conference on Object Oriented Programming Systems Languages and Applications, Anaheim, CA, USA, 2003.

[16] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg, "Model-Driven Development Using UML 2.0: Promises and Pitfalls," Computer, vol. 39, pp. 59-66, 2006.

[17] A. G. Kleppe, J. B. Warmer, and W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise: Addison-Wesley, 2003.

[18] J.-P. Tolvanen, "Domain-Specific Modeling: How to Start Defining Your Own Language," 2006, (Online) http://www.devx.com/enterprise/Article/30550 (Accessed 10 September 2008)

[19] S. Tang, M. Hanneghan, and A. El-Rhalibi, "Pedagogy Elements, Components and Structures for Serious Games Authoring Environment," presented at 5th International Game Design and Technology Workshop (GDTW 2007), Liverpool, UK, 2007.

[20] S. Tang and M. Hanneghan, "Educational Games Design: Model and Guidelines," presented at 3rd International Game Design and Technology Workshop (GDTW 2005), Liverpool, UK, 2005.

[21] S. Tang, M. Hanneghan, and A. E. Rhalibi, "Pedagogy Elements,Components and Structures for Serious Games Authoring Environment," presented at 5th International Game Design and Technology Workshop (GDTW 2007), Liverpool, UK, 2007.

[22] E. M. Clarke, J. M. Wing, and e. al., "Formal Methods: State of the Art and Future Directions," ACM Computing Surveys (CSUR), vol. 28, pp. 626-643, 1996.

[23] V. S. Alagar and K. Periyasamy, Specification of Software Systems: Springer, 1998.

[24] A. Bobbio, "System modelling with Petri nets," in Systems Reliability Assessment, A. G. Colombo and A. S. d. Bustamante, Eds. Netherlands: Springer, 1990, pp. 103-143.

[25] O. S. Noran, "UML vs. IDEF: An Ontology-oriented Comparative Study in View of Business Modelling," presented at 6th International Conference on Enterprise Information Systems, ICEIS 2004, Porto, Portugal, 2003.

[26] J. Erickson and K. Siau, "Can UML Be Simplified? Practitioner Use of UML in Seperate Domains," presented at Workshop on Exploring Modelling Methods for Systems Analaysis and Design, 2007.

[27] M. J. Taylor, M. Baskett, G. D. Hughes, and S. J. Wade, "Using Soft Systems Methodology for Computer Game Design," Systems Research and Behavioral Science, vol. 24, pp. 359-368, 2007.

[28] R. Hunicke, M. LeBlanc, and R. Zubek, "MDA: A Formal Approach to Game Design and Game Research," presented at Challenges in Game AI Workshop, AAAI04, 2004.

[29] J. Kienzle, A. Denault, and H. Vangheluwe, "Model-Based Design of Computer-Controlled Game Character Behavior " in Lecture Notes in Computer Science, vol. 4735/2007, G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, Eds.: Springer Berlin / Heidelberg, 2007, pp. 650-665.

[30] S. Tang, M. Hanneghan, and A. E. Rhalibi, "Modelling Dynamic Virtual Communities within Computer Games: A Viable System Modelling (VSM) Approach," presented at 4th International Game Design and Technology Workshop (GDTW'06), Liverpool, UK, 2006.

[31] T. Dalgleish, "The emotional brain," Nature Reviews Neuroscience, vol. 5, pp. 582-585, 2004.

View publication stats