A Platform Independent Game Technology Model for Model Driven Serious Games Development

Stephen Tang[1], Martin Hanneghan[2] and Christopher Carter[3]
Liverpool John Moores University, Liverpool, UK
o.t.tang@ljmu.ac.uk
m.b.hanneghan@ljmu.ac.uk
c.j.carter@ljmu.ac.uk

Abstract: Game-based learning (GBL) combines pedagogy and interactive entertainment to create a virtual learning environment in an effort to motivate and regain the interest of a new generation of 'digital native' learners. However, this approach is impeded by the limited availability of suitable 'serious' games and high-level design tools to enable domain experts to develop or customise serious games. Model Driven Engineering (MDE) goes some way to provide the techniques required to generate a wide variety of interoperable serious games software solutions whilst encapsulating and shielding the technicality of the full software development process. In this paper, we present our Game Technology Model (GTM) which models serious game software in a manner independent of any hardware or operating platform specifications for use in our Model Driven Serious Game Development Framework.

Keywords: game technology model, platform independent game technology model, serious games engineering, model driven engineering, games based learning, model driven serious games development

1. Introduction

Game-based learning (GBL) refers to both the innovative learning approach derived from the use of computer games that possess educational value and other software applications that use games for learning and education purposes (e.g. learning support; teaching enhancement; assessment and evaluation of learners etc.) (Tang, Hanneghan, & El-Rhalibi, 2009). These computer games, designed for non-entertainment purposes, are also widely known as serious games. The preliminary results (Jenkins, Klopfer, Squire, & Tan, 2003) of GBL are demonstrating great educational promise. As computer gaming becomes a digital culture deeply rooted amongst the new generation of learners, many researchers and practitioners agree it is now appropriate to exploit gaming technologies in order to create a new generation of educational technology tools to equip learners of all ages with necessary skills via GBL (FAS, 2006).

However, the absence of high-level authoring environments and support for non-technical domain experts (or teachers) to create custom serious games are impeding many who wish to adopt this innovative learning approach (Tang & Hanneghan, 2010). Despite efforts to create serious games through bespoke in-house development, using open source or royalty-free game engines in collaboration with a team of developers and 'modding' (or *modifying*) commercial-off-the-shelfgames by utilising a game editor application, the adoption rate of GBL in the mainstream is still low. Many of these tools and technology platforms for producing serious games are readily available but most of these tools require substantial knowledge in games development which hinders non-technical domain experts from adopting games-based learning.

Advancements in software engineering are making the creation of high-level serious games authoring environments for non-technical domain expert more accessible. The MDE (Model Driven Engineering) approach allows aspects of serious game software to be represented formally as an abstract model which can be automatically transformed into more refined software artefacts and subsequently into serious game software applications. This approach enables non-technical domain experts to produce serious games for the use of GBL easily and quickly (and possibly at a lower cost) through Domain Specific Modelling Languages (DSML) and therefore lowers the barriers that hinder the production of these applications. We believe that MDE can help to facilitate the adoption of GBL.

In this paper, we present our platform independent Game Technology Model for use in our Model-Driven Serious Game Development Framework. Aspects of serious games represented in our Game Content Model (Tang & Hanneghan, 2011) will be translated into the Game Technology Model where additional computational information are added into this refined model. In the next section, we describe MDE and how this approach can benefit serious game development for game-based

Reference this paper as: Tang, S, Hanneghan, M and Carter, C. "A Platform Independent Game Technology Model for Model Driven Serious Games Development" *The Electronic Journal of e-Learning* Volume 11 Issue 1 2013, (pp61-79), available online at www.ejel.org

learning. We then revisit our Model-Driven Serious Games Development Framework before we review in Section 3 and analyse a selection of relevant game engines or game software frameworks in Section 4. Consequently, we identify core technologies used in the creation of game software in Section 5. In Section 6, we detail our Game Technology Model and describe this model in relation to our existing Game Content Model. In addition, we explain the application of Game Technology Model in our model-driven serious games development approach. Finally, we conclude the paper with a brief discussion on Game Technology Model in Section 7 and outline future work in this exciting and promising research area in Section 8.

2. Model driven engineering and serious games development for game-based learning

Model-Driven Engineering (MDE) refers to a software development approach that relies extensively on the use of graphical or logical models to represent aspects of software and automates the transformation of models into more refined software artefacts. Models are primary artefacts in the development process expressed using a Domain Specific Modelling Language (DSML) to formally represent the structure, behaviour and requirements of a particular domain. Aspects of models are analysed and translated through transformation engines and generators to synthesize software artefacts consistent with these models. This approach can help in alleviating the platform complexity and expressing domain concepts effectively (Schmidt, 2006).

MDE is generally domain specific and only suitable for developing software specific to a targeted platform with the aim to automate manual coding. Expertise of programmers is embedded into the underlying framework and code generators allowing domain experts to provide complete specifications of the problem domain without worrying about the technical aspects of software development (Kelly & Tolvanen, 2008).

At the core of MDE is the model. A model is defined as “a simplification of a system built with an intended goal in mind [that] should be able to answer questions in place of the actual system” (Bézivin & Gerbé, 2001). Technically a model is described as a set of statements that effectively describes a *system-under-study* (SUS) (Seidewitz, 2003). Effectively, a model is a graphical or logical representation of a SUS (Favre & Nguyen, 2005). In a software context, models are used as a form of conceptual representation of a software system to facilitate the production of concrete software (Bézivin, 2004). In a model-driven approach, the model is used to assist the system modeller to more accurately describe the SUS. The model is expressed by the system modeller through statements, which are expressions that hold truths about the SUS. The validity of the model of the SUS is dependent on the correctness and completeness of statements describing it. Statements can be expressed informally through natural language, but are best constructed using formal notations which adhere to the grammar of the formal language. Such formal languages are generally termed as modelling languages or Domain-Specific Modelling Language (DSML). In the context of game-based learning, MDE can provide the environment for domain experts to produce a serious game via modelling (either using language or visual tools) without worrying about the intricacy of game development. We believe that this approach can help to simplify the route to serious games production and therefore ease the adoption of game-based learning in mainstream education and training.

MDE notably promises great benefits to its practitioners. From a software development context, MDE offers an increase in productivity, promotion of interoperability and portability among different technology platforms, support for generation of documentation, and easier software maintenance (Kleppe, Warmer, & Bast, 2003). In addition, it can also lead to production of better code quality and reliability due to integration of domain rules into the DSML which minimises modelling error and increases the reliability of mapping from model to code (Kelly & Tolvanen, 2008). From the perspective of the non-technical domain experts’ who wish to produce serious games, the MDE’s ability to encapsulate technical aspects of development via a DSML massively lowers the barriers that hinder the production of serious games for use in game-based learning.

3. A model driven serious games development framework

Our Model-Driven Serious Game Framework (featured in Figure 1) consists of nine parts namely: (1) User Interfaces (UI), (2) Models, (3) MDE Tools, (4) Components Library, (5) Code Templates, (6) Artefacts, (7) Technology Platform, (8) Operating Platform and (9) Software. The loosely coupled configuration allows framework developers to flexibly substitute modules whilst maintaining the

integrity of relationships among the modules via well-defined interfaces. It also clearly divides the views of entities whilst promoting a structured and systematic workflow (Tang & Hanneghan, 2010).

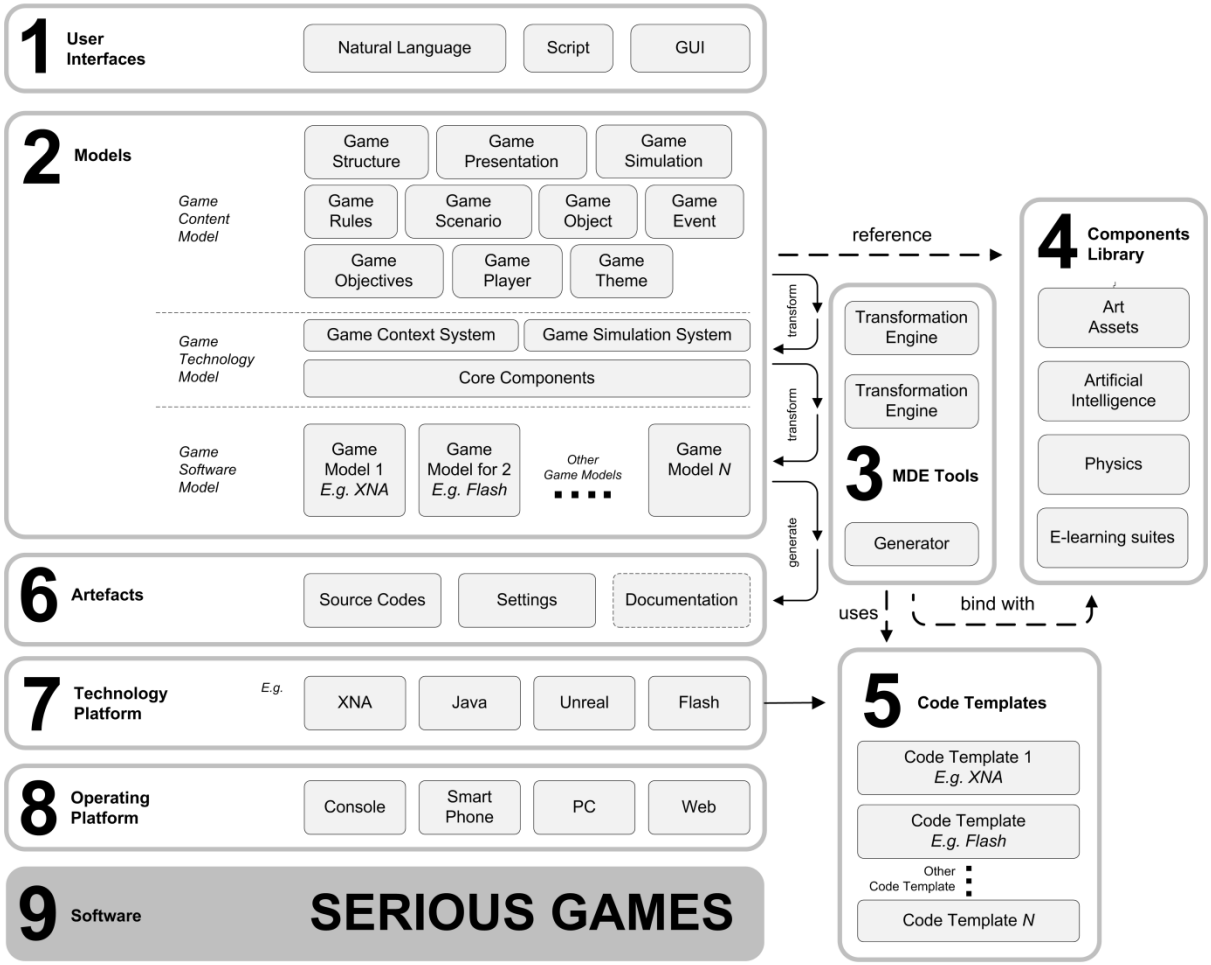

Figure 1: Model-driven serious games framework (Tang & Hanneghan, 2010).

At the core of this framework is a three-layer model abstraction that offers a higher level of encapsulation and greater interoperability support for serious games. These models are: Game Content Model (GCM), Game Technology Model (GTM) and Game Software Model (GSM) respectively. The Game Content Model represents the logical design specification of a serious game whilst the Game Technology Model is a computationally dependent model of the serious game but one which remains independent of technology or language platform. The Game Software Model, by definition, represents the transformed model of the serious game specific to a chosen technology platform. These models are then transformed into more refined artefacts using specific MDE tools (3) before generating the appropriate software artefacts. Transformation of models from one viewpoint to another is triggered automatically.

In our framework, the Game Technology Model transformation engine reads the Game Content Model and represents the game as a software model. The Game Technology Model is read by the Game Software Model transformation engine which reorganises the game as a software model compatible with a specific game software framework such as Unity, Unreal and XNA (or a proprietary built game software framework) by replacing game logical software constructs with the corresponding physical game software constructs. Finally, the Game Software Model is interpreted by a generator to compose software code (6) from predefined code templates (5) through mapping techniques. Game software framework constructs and code templates are defined by framework developer with reference to the specification of Game Software Model (see Figure 2).

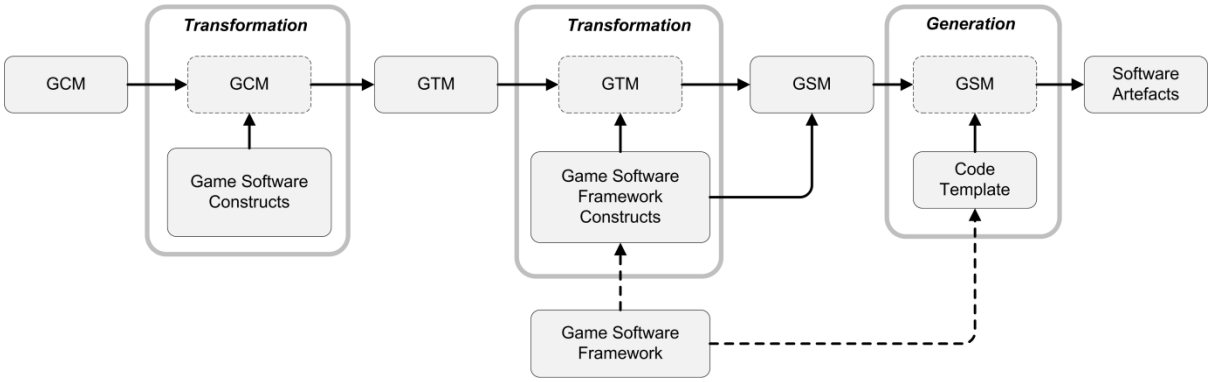

Figure 2: Transformation pipeline.

3.1 Game content model

The Game Content Model represents a game ontology from an interactive content viewpoint. It is used to document the design specification of a computer game and will be utilised as the master model for building other game models in our model-driven serious games development framework. Since we focus mainly on serious games, we have limited the current scope of our game ontology covers the game concepts used in the documentation of role-playing and simulation game genres because we believe they are the most suitable for utilisation within the context of education and training compared to other game genres but the nature of this ontology is that is can support further concepts in the future with ease.

Our Game Content Model improves upon on the existing work of the Game Ontology Project (GOP) (Zagal, Mateas, Fernández-Vara, Hochhalter, & Lichti, 2005), Rapid Analysis Method (RAM) (Järvinen, 2007) and the Narrative, Entertainment, Simulation and Interaction (NESI) model (Sarinho & Apolinário, 2008). Whilst there is a degree of commonality with these models in definition aspects of games with these models, they lack the formalism required for model translation and are devoid of the concepts that are essential in describing characteristics of a game from a software engineering perspective. We have selectively combined these with our study on game design, game development and serious games in order to introduce additional concepts required to represent the game, organising these concepts into a meaningful object-oriented structure which helps us during the translation process using MDE tools later. The topmost level of our game content model consist of ten interrelated key concepts that best represent the rules, play and aesthetic information of a computer game and they are *Game Structure, Game Presentation, Game Simulation, Game Rules, Game Scenario, Game Event, Game Objective, Game Object, Game Player* and *Game Theme* (Tang & Hanneghan, 2011).

The game structure provides the form and organises the game into segments of linked game presentations and game simulations. In our framework, we refer game presentation as a medium for presenting game information which is composed of media and Graphical User Interface (GUI) components. Whereas game simulation is defined as an environment which simulates a game scenario in accordance to the definition of game rules and game physics. The interactions between game objects and the results of an interaction in a game simulation are defined using game rules. A game simulation can be used to host multiple game scenarios aligned with the storyline. Each game scenario is setup using a selection of game objects to create an environment, a sequence of game events and a set of game objectives that challenges both player skills and knowledge of the game. The game player can control game object(s) and interact via hardware or graphical user interface controls. And finally, the game theme describes the “look and feel” of the game. The relationships between these key concepts are illustrated in Figure 3.

In our model driven approach, the formal description of a serious game, compliant to our Game Content Model, is subsequently translated into a computational representation independent of operating platform: the Game Technology Model. The remainder of this paper details the workings of our Game Technology Model for use in our Model-Driven Serious Games Development Framework.

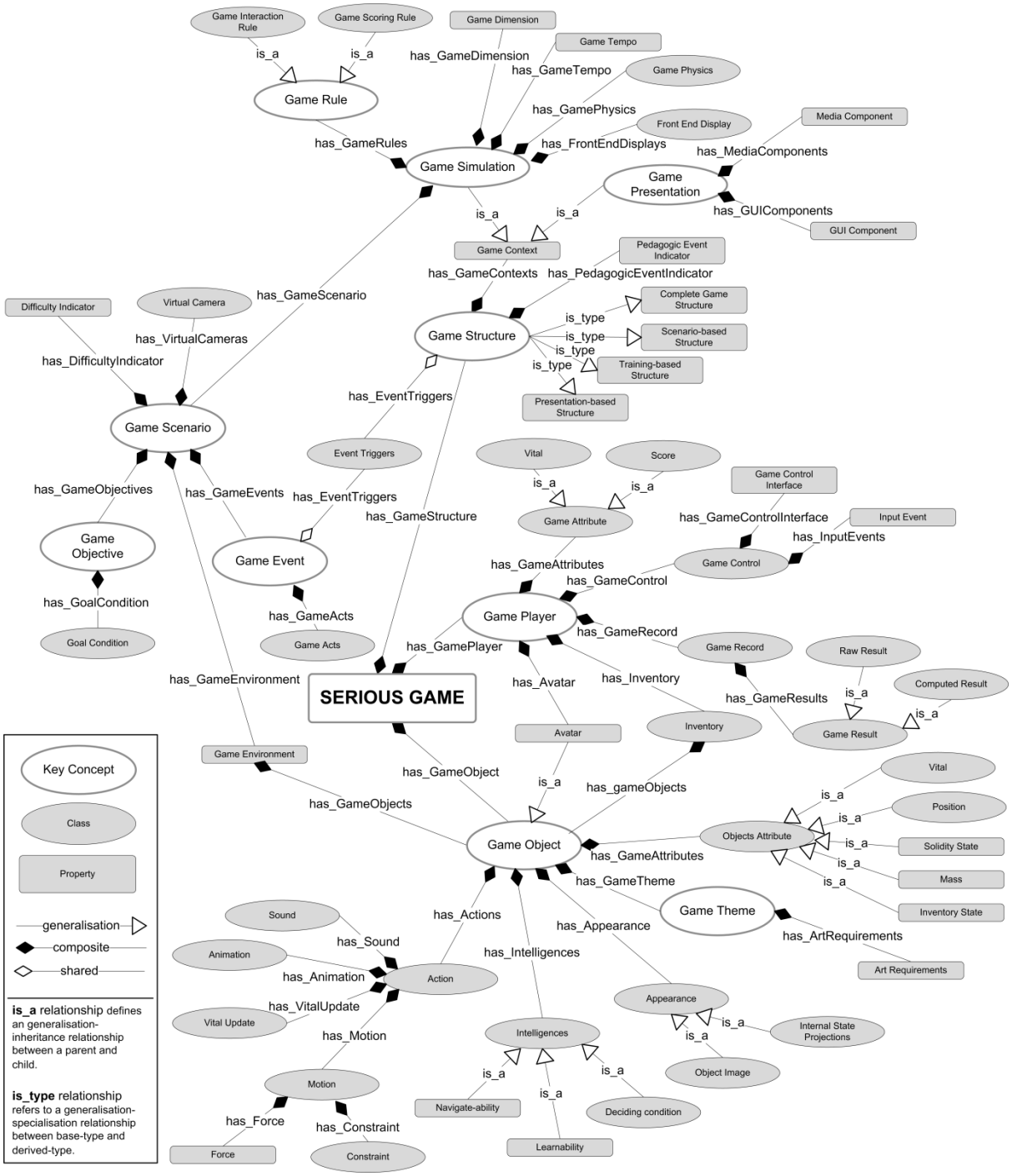

Figure 3: Overview of the game content model (Tang & Hanneghan, 2011)

4. Game engine analysis

In the past, games software were written as singular entities in assembly languages that were tightly coupled with the underlying hardware platform to utilise hardware resources efficiently and hence provide a seamless gaming experience (Bishop, Eberly, Whitted, Finch, & Shantz, 1998). However, this approach permits little code reuse and low code scalability which results in complex games being both costly and timely to develop. Coding these games is also a highly specialised skill.

A game engine or game software framework is the technical and economic solution for writing modern, complex games. It consists of subsystems or software components that perform a number of distinct tasks (such as graphic rendering (2D or 3D); game physics computation such as collision detection; collision reaction and locomotion; programmed intelligence; user input; game data management and other supporting technologies) to operate the game software. These software

components are built to manage, accept, compute and communicate data with other software components without fail. Not all game engines support the entire feature set required for all game genres, since integrating these technological components under a single framework would be a prohibitive task. Furthermore, not all computer games software will require the entire collection of software components to function.

The early streamlined approach to game engine architecture (illustrated in Figure 4) were composed of the software components that handles input, audio, graphic representation and the dynamics (game mechanics) (Bishop, et al., 1998).

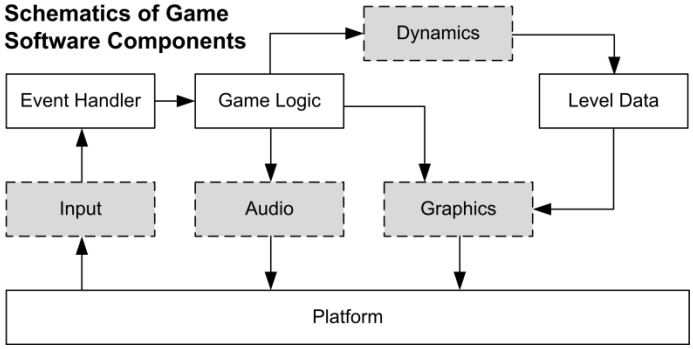

Figure 4: Aspects of game software illustrated in shaded rectangles are elements of a game engine while game logic and level data are regarded as *content* that defines a game (Bishop, et al., 1998)

Modern day game engines are capable of computing complex 3D scenes with dynamic objects; rendering realistic graphics; planning and deciding actions for non-player characters (NPC); and supporting multiple players concurrently over a network. The architecture of the Delta3D engine documented by Draken, McDowell & Johnson (2005) in Figure 5 exhibits the addition of software components such as character animation, scene graph and networking to assist the development of modern 3D games with multi-player support.

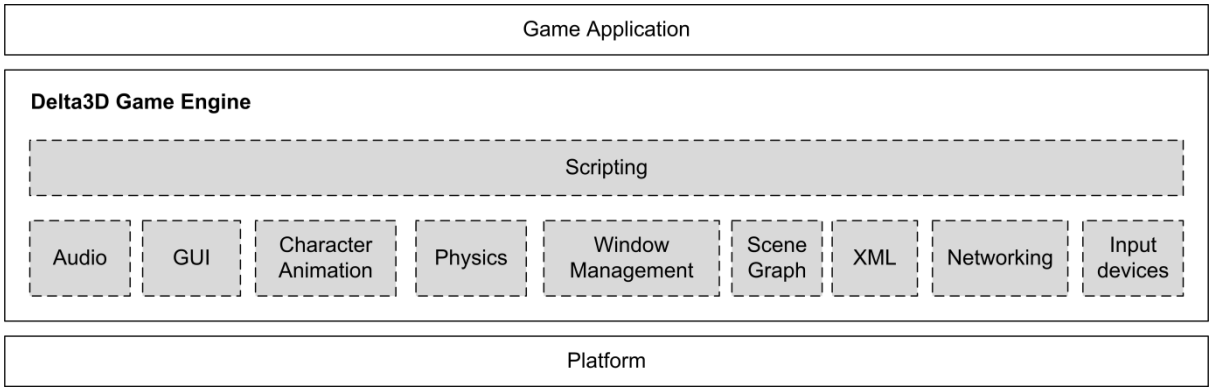

Figure 5: Architecture of the Delta3D game engine (Darken, et al., 2005).

Current generation commercial game engines such as Unreal Engine, CryEngine and EgoEngine are packed with grander features and sophisticated tools that enable creation of high quality game software. The game engine architecture explained by Gregory (2009) (see Figure 6) illustrates the typical logical architecture of a modern 3D game engine. It consists of layers abstract to game specific software components which are wired for maximum reusability (software components are shown shaded) and interoperability across different platforms (through the Platform Independence Layer). The Game-Specific Subsystems and Gameplay Foundations are components that can be re-written to adapt other components for other game genres with minimal changes to the remaining software components (some changes are still necessary as these components may be written specifically for the chosen game genre for optimal performance and reliability).

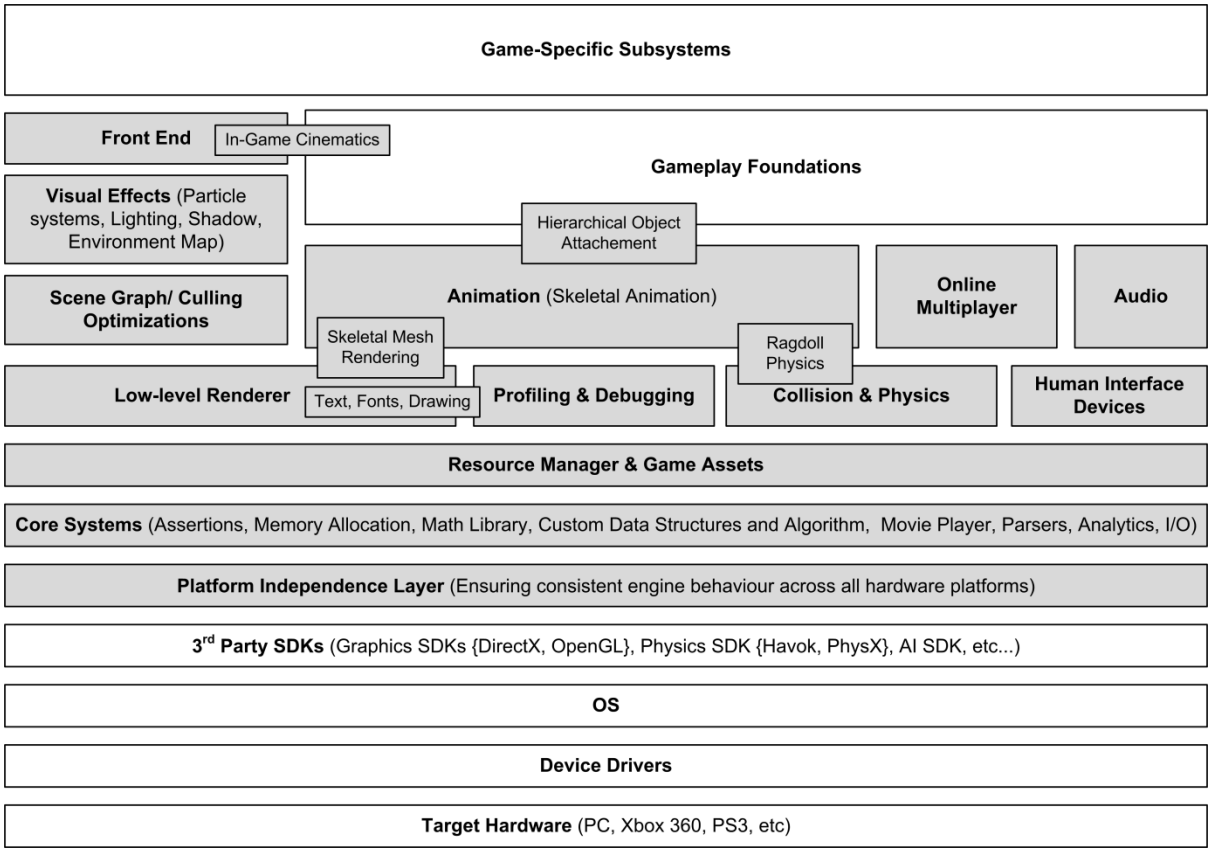

Figure 6: Overview of commercial grade game engine architecture (Gregory, 2009).

In this study, we are interested in the fundamental game engine technologies abstracted from any particular game genre. We raise the following questions: What are the core technologies needed to run a serious game? How can we best represent these game technologies in a model for use in a model-driven approach? In the remainder of this paper, we address these research questions.

5. Core technologies for game software

Computer games are typically structured to consist of user interface components such as menus for game configuration and selection of game scenarios, and the simulation of a scenario (game level) which demands the game player's interaction with the system. It is common at the start of a scenario to have cut-scenes to present the story that drives the game as a precursor for immersion and motivation building. Within the simulation of a scenario, the game software cyclically performs a series of tasks successively using a time step of typically 30-60 iterations (frames) per second. These tasks typically include handling input from processing network data from other clients (in multiplayer games), human interface devices and Graphical User Interface (GUI), simulating motion and artificial behaviour, checking for collisions, updating the internal state of game objects, rendering visuals on the screen and processing other tasks such as audio and sending out data packets (Llopis, 2005).

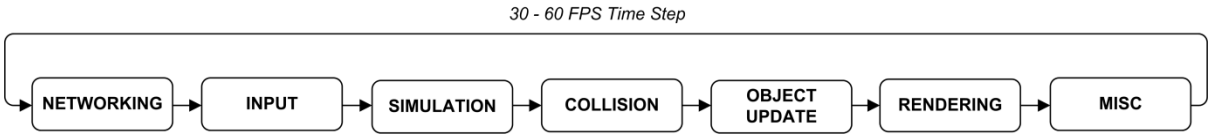

Figure 7: Tasks in game software (Main Game Loop)

Components of game engines vary depending on technology features and are often constrained by particular game genres, technology platforms (hardware) and visual dimension of the game world (2D or 3D), but there are some common technology components across all game engines. These are the core technologies that enable the creation of a variety of game software solutions, each featuring creative and compelling content. The core technologies we identified are *graphics (renderer), animation, audio, input, game physics, user interfaces, networking* and *game resources management*. In the following subsections we describe these core technologies in brief.

5.1 Renderer

The renderer is key component to any 2D or 3D graphic engine that is responsible for graphics-related computations and rasterised screen output. The 2D renderer provides the interface for graphic hardware and draws 2D graphics, whilst the 3D renderer provides additional functionalities such as loading 3D models, rendering and managing textures, applying different type of materials and blends to the texture, rendering static and dynamic lighting in the scene, displaying viewports and virtual screens, and providing control to the virtual camera. Other features which are found in a renderer also include particle systems and post-processing used for visual effects. In recent years, 3D graphic engines have gained significant popularity over 2D graphic engines due to the consumer demand for 3D games, whilst most 3D engines also support creation of 2D games through some clever exploitation of the technology, such as orthographic projection (Gregory, 2009). However, 2D graphic engines are still relevant, particularly on lightweight platforms such as the mobile and web platforms.

5.2 Animation

The animation component is responsible for determining the next pose of a 3D model or the next frame of a sprite, independent of its spatial position within the world. A 2D animation component would extend the functionality of a 2D graphic engine to include management of animation state which is why many would regard the animation subsystem to be part of the renderer. However, the animation data is different from graphic data and therefore it should be processed separately by the animation component. The 3D animation component for 3D is conceptually similar to the 2D setup, retrieving animation data from a file in order to modify the skeleton or vertex mesh of a 3D model (i.e. Skeletal vs. Morph Target animation (Gregory, 2009)). Advanced facilities such as blending (transitioning from one animation state to another), inverse kinematics (IK), decompression of animation data, animation playback and procedurally animating free-form body movement are also packaged within this component.

5.3 Audio

The audio component offers facilities to interface with the audio hardware and manage the playback of audio. In a 3D game engine, the audio subsystem is extended to include 3D audio model which allow game players to perceive sound originating from a positional source.

5.4 Input

The input component handles all the input events triggered by game players either via traditional human interface devices (HID) such as keyboard, mouse, joystick or newer gestural devices such as the PlayStation® Move and Microsoft® Kinect. Each input event is managed by the input subsystem to trigger specific instructions including GUI events and in-game commands.

5.5 Game physics

Game physics is the component that applies the law of physics making the game world to behave realistically. Motion, collision detection and collision reaction are the generic forms of computation performed by the game physics component. The complexity and scope of game physics are dependent on game genre and type of game. For example, games such as *Gran Turismo* (http://www.gran-turismo.com/) would expect the physics computation to be very realistic whereas the game physics in *Need for Speed* (http://www.needforspeed.com) is more forgiving and tuned to accommodate the game-play.

5.6 User interfaces (UI): graphical user interface (GUI), media and heads-up display (HUD)

Game software uses GUIs both in-game and out-of-game as a graphical means for accessing functions or commands within the system. Buttons, list-boxes, checkboxs, radio-buttons, textboxs, tabs and scrollbars are each examples of commonly used GUI components. For critical game information that affects game-play, HUD elements such as a digital counters, analogue gauges, mini maps and horizontal graphical bars are also often used to notify game players about the changes of game state and so aid in the game-playing process.

5.7 Video player

Media such as text, graphics, sound and video are also widely used in games for presenting game related information. The intricate process of video playback is carried out by the video player component. Video playback is often used in game to show cinematic cut-scenes and perform the majority of the storytelling (some games may opt to use scripted animation using the animation component to provide seamless continuity from cinematic to game-play).

5.8 Game resources management

Game resources or game assets are usually produced using Digital Content Creation (DCC) tools. These can include 3D models, textures, materials, fonts, 2D sprite sheets, collision data, animation data, sound files, level data and others. The game resource management is the component that provides the facilities for loading and unloading these resources into the game system.

5.9 Artificial intelligence (AI)

AI provides Non-Player Character (NPC) abilities to decide, plan and act in a game scenario. Commonly used AI techniques in games include goal-driven decision making, path finding and perception traces which are adjusted to provide a challenging and yet balanced form of game-play. Again, these techniques can vary depending on games genre. Real-time strategy games such as *Command and Conquer* (http://www.commandandconquer.com/) use a wide range of AI techniques such as chasing and evading, flocking, path-finding and case-based reasoning, whereas games such as *Need for Speed* would rely heavily on waypoint navigation.

5.10 Networking

The networking component is used in games software to facilitate multi-players. It handles all the aspects of communication between client and server. Elements of networking component include authorisation, authentication, structuring and filtration of game messages, low level communications, task interface, protocols, network introspection, and data interpolation and extrapolation (Chris Carter, Rhalibi, Merabti, & Bendiab, 2010). This component is excluded from use when game software is developed solely for single player on local machine.

6. Game technology model (GTM)

Game technologies used to be closely coupled with a given hardware platform. As practice in software engineering advances towards the trend of interoperable software code, the game industry is adopting this approach to allow rapid release of game titles on multiple platforms through the introduction of a platform independence layer, as seen in the game engine architecture diagram in Figure 6. The responsibility of platform independence layer is to ensure that all components in the game engine behave consistently across different hardware platforms. This is similar to the functionality of a platform independent model from a model viewpoint.

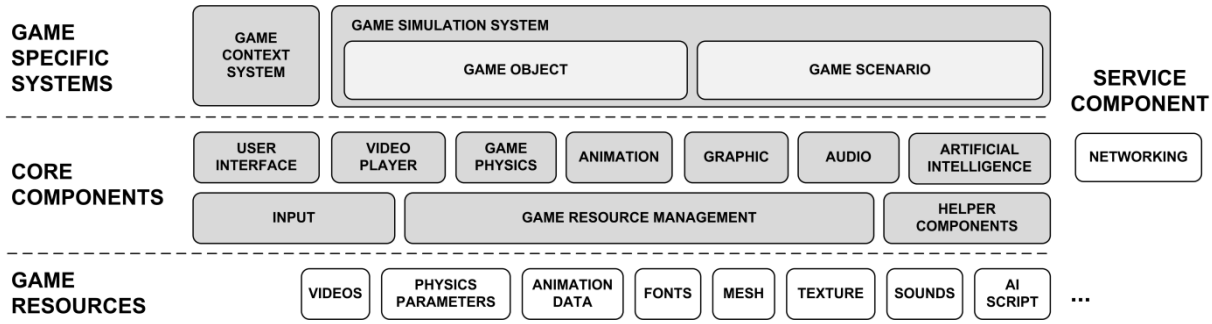

Figure 8: Overview of game technology model

The Game Technology Model in our model-driven framework represents serious games in a manner that is computationally independent of any operating platform. This is responsible for ensuring all components in the game engine behave consistently across different hardware platforms. Our proposed Game Technology Model in Figure 8 features two primary layers: the *Game Specific Systems* layer and the *Core Components* layer. The Game Specific Systems layer consists primarily of the *Game Context System,* which sets up the game and manages dynamic switching between the presentation context and simulation context, and the *Game Simulation System,* which populates the

world with game objects (both static and dynamic) and triggers the game events that form part of the game-play within a particular game scenario. It uses facilities provided by the core components layer to enable smooth running of both the Game Context System and Game Simulation System. Our analysis of game engines in Section 4 and the study into game processes and its key components have influenced the selection of core components in our GTM. We have selected *User Interface, Video Player, Game Physics, Animation, Graphic, Audio, Artificial Intelligence, Input, Networking* and *Game Resource Management* as core technologies in our Game Technology Model's core components layer. In addition to the core components, other components which are commonly used and deemed useful in game software includes the math library, random number generator and unique object identifier management. We have chosen to set aside networking as a service component instead of being a core component which developers can choose to use when developing multiplayer games. In our Game Technology Model, all the network data are treated as streams of input and this is handled by the message handler which is described in the networking component. In addition to the core components, other components which are commonly used and deem useful in game software includes the math library, random number generator and unique object identifier management. Although some of these components are readily built-in into certain technology platforms, this must be included in the Game Technology Model in order to achieve platform independence.

6.1 Game context system

Looking deeper into our Game Technology Model (see Figure 9), the responsibilities of Game Context System are to set up the application by initialising the input hardware, graphic hardware and audio hardware, specifying access paths for resources, switching between different contexts and freeing up hardware resources prior to shutting down the game application. In our Game Content Model, we break down a game into sections known as *game context*. A game context describes the type of game content to be presented to game players which can be either in the form of a *game presentation* or a *game simulation* (Tang & Hanneghan, 2011). We regard game scenarios as content to a game simulation. Separating game scenario from game simulation enables us to use the same game scenario in a different yet compatible game simulation setup. These contexts are loaded into the *Game Context System's Active Contexts Stack* based on the flow of the game defined. This stack-based approach allows multiple contexts to be rendered in the correct order, producing a layered effect as described in (C. Carter, El Rhalibi, Merabti, & Price, 2009).

Each context has its own methods for: initialising the data or components required; an update method which updates state of the context; and a render method which presents the visuals on the screen. Update and render routines are invoked at 30-60 time steps per second. The updates can also be multithreaded using fork-join parallelism (Kim & Agrawala, 1989). The clean-up method is invoked when a context is popped out from the active contexts stack to free off resources. Game contexts which are frequently used can be placed in a resource pool to avoid constant loading and unloading which could result to performance slowdown.

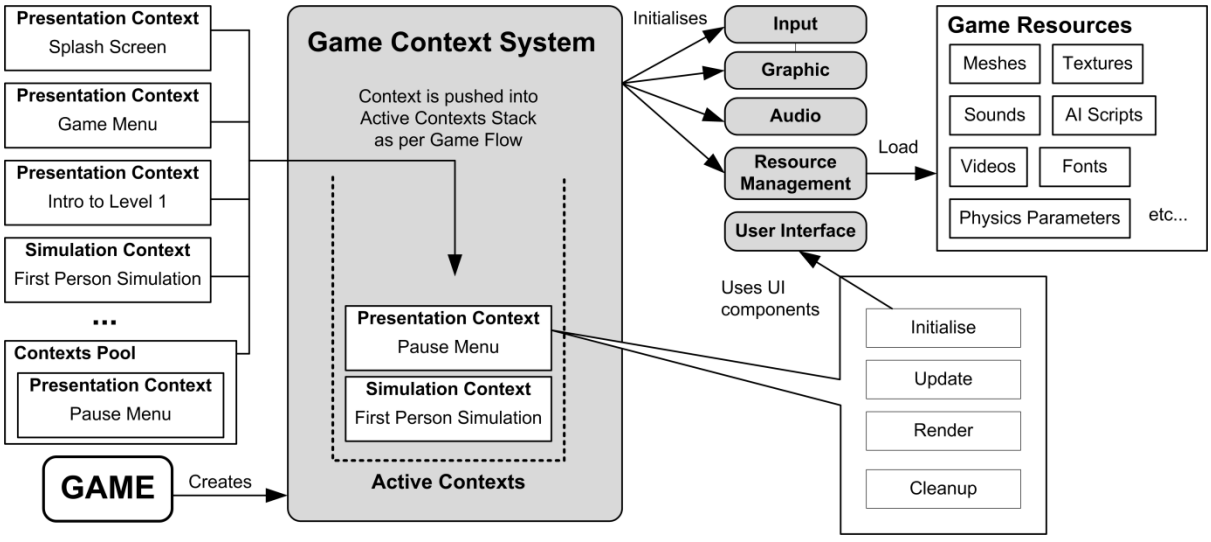

Figure 9: Game context system

The transition from one context to another is triggered by the event trigger. An event trigger monitors a defined event which can be an application event (a range of game application related of events such media event (such as “onMediaEnd”) and simulation events (such as simulation event “onSimulationEnd”, “onSimulationPause” and “onObjectiveUpdate”)), an input event (user input detected via hardware interface or graphical user interface (GUI)) or a time-based event. When the condition of the event is met, it notifies the Context Manager to push out the expired context and pop in the next context into the Active Context stack. Each event trigger is synchronised with the main update method which can be monitored by a manager such as the Game Mechanic Trigger and the Input Trigger whereas Time Trigger and proximity trigger are monitored in the respective update methods (which are also synced with the main update method) of the class it composed of.

At the top-most level of the game software is the serious game application itself which uses the game context system and provides the necessary interfaces for the game player to interact with the game. The game player component as described in the Game Content Model is composed of a reference to a game object which acts as an avatar, a set of attributes that determines the game player’s vitality or performance, a set of control maps which wires input events to the associated action of the avatar, a structure to store virtual items that player can own in the game and a statistical log of the player’s performance which are integrated into the game.

A game can be dependent on a fixed time step or variable time step. In our Game Technology Model, we favour for simplicity over complexity and hence has opted for fixed time step implementation. Within the game update method it checks for any input and application events and synchronises with the update method for each context.

6.2 Game simulation system

When an instance of a simulation context is made active, the Game Simulation System loads in the associated description of game scenario. A game scenario, in our model, is represented by a game environment (that is composed of a collection of game objects), a set of game events, a set of virtual cameras, a difficulty indicator and a set of game rules (which dictates the outcome of the interaction from two game objects) (Tang & Hanneghan, 2011). Game objects required for construction of the game environment are created and organised in a scene graph allowing the rendering of graphic components in the correct order. Data associated to game objects are stored in a collection which can be fetched and updated directly rather than having to traverse through the scene graph.

The Game Simulation System maintains two scene graphs; a scene graph for media, GUI and HUD (2DGraph scene graph); and a scene graph for proximity triggers and game objects which make up the game environment. The scene graph that stores the game environment will be rendered to the image buffer before the 2D scene graph to allow final construction of the render frame which has the media, GUI and HUD overlaid on top of the game environment.

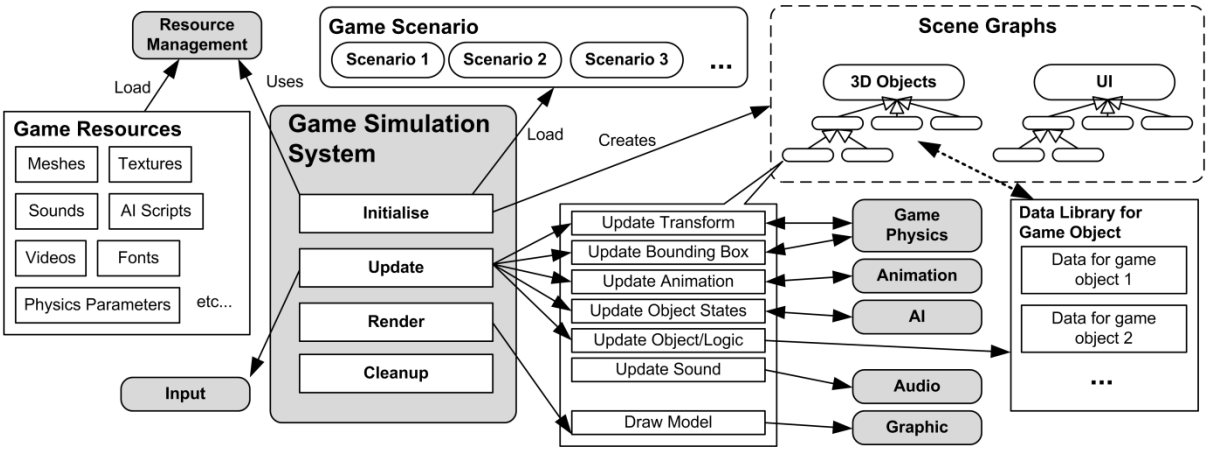

Figure 10: Game simulation system

At each update routine of the Game Simulation System, all game objects have their transform and animation updated. Object state, game data and other computationally demanding process such as collision check of the game objects are updated at different time intervals to avoid performance slowdown. Dynamic objects which no longer exist are removed from scene graph and data associated

are destroyed during runtime. The events defined in the game scenario are also triggered in the update routine of the Game Simulation System. All update and render routines are in synchronization with the main game loop.

6.2.1 Game scenario

A game scenario holds all the relevant data and data structures that represent a level in a game. Most of the description of a game scenario can be represented as data but descriptions such as game interaction rule and game scoring rule are automatically embedded in the respective game object *Update()* method whereas events are encoded in the game scenario *Update()* method for optimal performance using our MDE tool. In addition to triggering game events, the *Update()* method in a game scenario checks if all game objectives are met and updates the game objects within the game environment scene graph.

6.2.2 Game object

A game object is the primary data structure which will be processed in the Game Simulation System using the facilities provided by the game component. Game objects consist of attributes that hold value of existence such as objects attributes, position, mass, solidity state and size of inventory (which defines the amount of objects associated to itself), and behavioural characteristics which are represented in the form of motion, action, attribute updates, sound and intelligence. These can be represented into three distinct classes namely Actor, DynamicObject and StaticObjects.

In terms of mapping the game object from Game Content Model to Game Technology Model, the object attributes and animation sequences are easily mapped from the definition in Game Content Model into the structure. Action definitions of the game object such as animation, sound, motion can be invoked from the Update() method. These are grouped into the relevant action state of the game object as defined in the Game Content Model. This approach allows easy pairing with input event or decision making in AI. The input or AI components will only need to change the *activeState* of the object to trigger the paired action. Each action is marked in a conditional statement using a unique identifier as illustrated in Figure 11. At each game object update call, it checks the *activeState* which is changed when certain input event is triggered or AI decisions are called. Input events are updated at the main game loop whereas AI routine is done right before the invocation of the actions.

```
If(this.actionState == 2)//Pick
{
      collisionObject = PhysicManager.checkCollision(this, GameEnvironment)
      If(collisionObject.type == "HealthPack")
      {
            AnimationManager.SetSequence(this, "Pick")
            this.health.add(25.0f)//health is a vital
      }
      ElseIf(collisionObject.type == "Key")
      {
            AnimationManager.SetSequence(this, "Pick")
            this.inventory.add(object);
      }
}
```

Figure 11: Example of action definition in a game object

Within each action, further query can be invoked to determine the actual state of an object. This could be in the form of collision checking to determine if a game object is within the boundary of another game object (which is defined by a game interaction rule in Game Content Model) or checking an attribute's value or inventory of the game object as illustrated in the pseudo-code in Figure 11. An action, as described in the Game Content Model, consists of method calls to the relevant components such as physics and animation to update and transform the game object which is automatically transformed from specification of game design in our framework. It also consists of commands to invoke playback of audio files and relevant data updates to vital and inventory. The main update call will traverse the game environment scene graph and update all the states, data and transforms before executing the render routine.

The *Render()* method of a game object processes and displays the object appearance on the screen using facilities in the renderer component. In conventional approaches, all transforms are computed prior to the actual rendering process. In our data-driven approach, all the transform and rendering is computed by the renderer through an overloaded *Render()* method. This approach separates game logic from component computation simplifying transformation and generation of code.

6.3 Representing the game technology model in a model driven framework

In our framework, we favour the use of open data format such as eXtensible Markup Language (XML). XML is a specification for defining how information is stored (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2008) like any other earlier form of initiatives such as the CASE Data Interchange Format (CDIF) (Flatscher, 2002) and Portable Common Tool Environment (PCTE) (ECMA, 1997) introduced for use to store software engineering data. It offers great flexibility for defining the data format for representing models. In addition, XML can easily accept additional information from the automated transformation process between the models for MDE. Furthermore it is also well supported by MDE technologies such as Eclipse Modelling Framework (EMF) and Generic Modelling Environment (GME) (Ledeczi et al., 2001) making it the ideal choice for representing data-model. This makes XML is a viable option for defining data-model in our model driven framework.

6.4 Transforming the game content model into the game technology model

In our model driven serious game development framework, the Game Content Model is generated by our software tool, SeGMEnt (Serious Games Modelling Environment) (see Figure 12 and Figure 13 for screenshots of SeGMEnt). SeGMEnt is a web-based serious games modelling environment implemented using Adobe Flash. It is designed to allow non-technical domain experts to document serious games components either by entering required data through a data entry dialogue or via a visualisation environment for those aspects that involves positioning of in-game components in the virtual world, construction of an environment and layout of GUI components on-screen. It encapsulates the concepts of Game Content Model into the different design viewpoints namely structure, object, simulation, presentation, environment and player using the appropriate UI model. The Game Content Model generated from SeGMEnt is represented in XML format.

The Game Content Model then undergoes a transformation to be translated into the Game Technology Model, a computational model independent of platform, using a MDE tool. The MDE tool can be developed using existing MDE technologies such as EMF and GME as described in (Tang & Hanneghan, 2012) or implemented using any programming languages with XML parser facility. In our model-driven serious games development framework, we have developed a custom transformation tool in PHP. The transformation of Game Content Model to Game Technology is mainly a process of refining data and reformatting it into a computation independent model by reorganisation of data into programmatic structures. This also involves the addition of programmatic statement calls to the relevant Game Technology Model component's function to process the relevant data. There are two approaches to have a Game Content Model translated to a Game Technology Model. The first approach is to interpret the source model (Game Content Model) and then weave the additional information into the source model in order to produce the target model (Game Content Model). This approach would require the source model to be well structured and the translation process is just merely locating the token of information and weaving in the additional information into the source model to create the target model. The second approach is to traverse through the entire source model to locate the required token of information and a new target model is composed by structurally reformatting data in the source model and adding in the additional information. In our case, we opted for second approach because it does not constraint us to the structure of the source model.

The process of model translation from Game Content Model to Game Technology Model as described earlier requires (1) traversing the XML document structure in search for the marked elements which will be (2) reformatted and may include more information to construct the new model or artefacts. In PHP, there are various approaches for traversing XML document structure and this includes the Simple XML (core library in PHP 5.0), XML Expat parser and XML DOM (Document Object Model). In our prototype, we decided to use the Simple XML approach mainly because it is a simpler approach to traverse a document structure. This validates the notion that creation of a proprietary model translator and a code generator can be simple. In our approach, we specify the path of the XML structure to locate the marked data. Once marked data has been located, data can be accessed and

re-marked with additional information. Whenever there is more than one child node in the structure, the `foreach()` construct is used to iterate over the tree structure as shown in Figure 14.

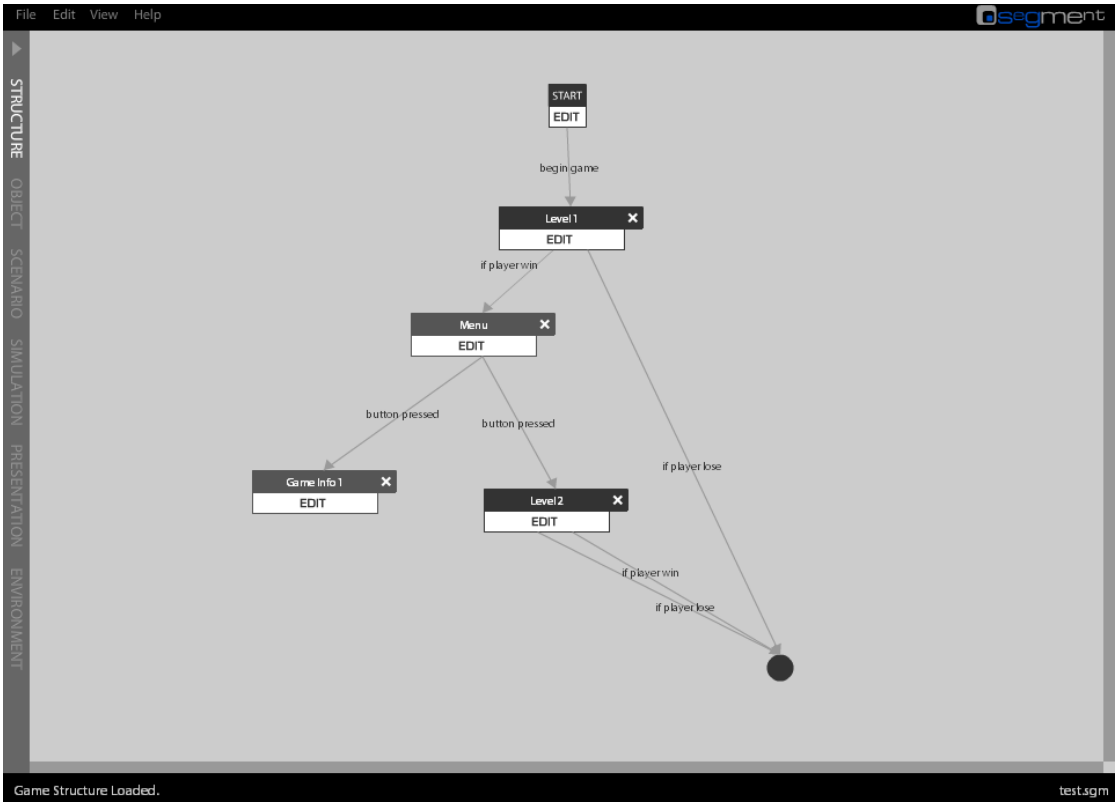

Figure 12: Modelling game structure in structure designer within SeGMEnt

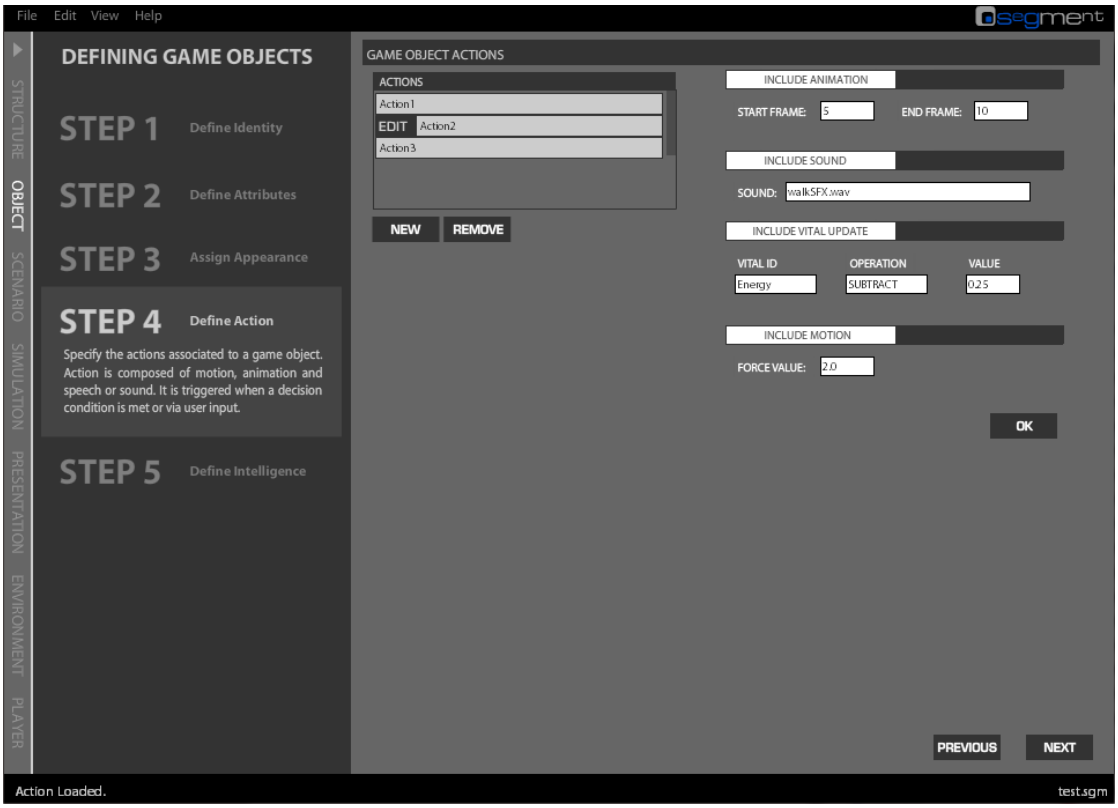

Figure 13: Modelling game object in object designer within SeGMEnt.


```
<%php
//Loading XML doc
$xmlDoc = simplexml_load_file("seriousgame.xml");

////More implementations… (omitted in this example)

//Game Player
$GTM_XML = $GTM_XML . "<gamePlayer>";
$GTM_XML = $GTM_XML . "<attributes>";
$gameAttribute = $xmlDoc->gameplayer->gameattributes->gameattribute;
foreach($gameAttribute as $ga)
{
   $GTM_XML = $GTM_XML . "<" . $ga->name ."StartValue.setValue>";
   $GTM_XML = $GTM_XML . "<constant value=\"" . $ga->startvalue . "\"/>";
   $GTM_XML = $GTM_XML . "<" . $ga->name ."StartValue.setValue/>";

   $GTM_XML = $GTM_XML . "<" . $ga->name ."EndValue.setValue>";
   $GTM_XML = $GTM_XML . "<constant value=\"" . $ga->endvalue . "\"/>";
   $GTM_XML = $GTM_XML . "<" . $ga->name ."EndValue.setValue/>";
}

$GTM_XML = $GTM_XML . "</attributes>";
$GTM_XML = $GTM_XML . "</gamePlayer>";

////More implementations… (omitted in this example)
%>
```

Figure 14: Snippet of code from the game technology model translator to locate a marked data and iterating through the tree structure

The new model or artefact is created by transforming data into a refined new format of data through the process of string concatenation. By using the string object, we can easily add in additional information and format data to the desired format. An example of Game Content Model generated from SeGMEnt is shown in Figure 15 and the Game Technology Model which is transformed using our proprietary tool is shown in Figure 16. The Game Technology Model will later be transformed into Game Software Model before it is used for generation of artefacts which can either be software code or a compiled software artefact. The transformation of Game Technology Model to Game Software Model is described briefly in Section 3 and will not be covered in depth because it is outside the scope of this paper.

7. Discussion

Our Game Technology Model is based on a data-driven architecture and includes the essential game specific systems and core components of software which facilitates the operation of serious games defined using Game Content Model. This data-driven architecture exhibits loose coupling allowing core technologies to be swapped and giving developers the flexibility to select the preferred components in order to support a particular genre of serious game. Although our Game Technology Model is currently designed to accommodate only simulation and role-playing genres of serious games, our Game Technology Model can also to be extended easily to support a wide range of genres by simply swapping the relevant core technologies with more specialised technologies.

The architecture of our Game Technology Model can be used by developers to create their own proprietary game software framework. Alternatively, the Game Technology Model can be regarded as a generic virtual wrapper for existing game software frameworks. The functionality of each component defined in the Game Technology Model act as interfaces that wrap a different implementation of a game technology. This allows serious games software to be produced on different technology platforms through code generation which reads the Game Technology Model and translates it into software artefacts.

In our model driven framework, the Game Technology Model is represented in XML to make marking and locating of marked information easier, and therefore simplifying the task of model transformation. Framework developers will need to have a deep understanding of the Game Content Model and Game Technology Model before they can transform the Game Content Model to the Game Technology Model. In our framework, we choose not to be constrained by the structure of Game Content Model and have opted to implement our MDE tool that locates the marked information in the Game Content Model and rebuilds the Game Technology Model from scratch. Development of the MDE transformation tool is proven to be much simpler and straight forward especially with modern XML programming interfaces such as Simple XML in PHP.

```
<gameObject>
   <id>fireman</id>
   <objectAttributes>
      <mass>75.0</mass>
      <solid>true</solid>
      <vitalAttribute>
         <id>health</id> <startValue>100.0</startValue>
<endValue>0.0</endValue>
      </vitalAttribute>
      <vitalAttribute>
         <id>energy</id> <startValue>100.0</startValue>
<endValue>0.0</endValue>
      </vitalAttribute>
   </objectAttributes>
   <objectAppearance>
      <spriteSource>asset/fireman/sprite.png</spriteSource>
      <spriteDimension>
        <height>20</height>
        <width>20</width>
      </spriteDimension>
   </objectAppearance>
   <objectAction>
      <id>idle</id>
      <animation>
         <startFrame>1</startFrame> <endFrame>3</endFrame>
      </animation>
   </objectAction>
   <objectAction>
      <id>walkLeft</id>
      <motion>
         <forceValue>5.0</forceValue> <forceAngle>-180</forceAngle>
      </motion>
      <animation>
         <startFrame>4</startFrame> <endFrame>8</endFrame>
      </animation>
      <vitalUpdate>
         <attributeID>energy</attributeID>
         <arithmeticOperator>-</arithmeticOperator>
         <constant>0.1</constant>
      </vitalUpdate>
   </objectAction>
</gameObject>
```

Figure 15: XML describing the fireman game object generated from SeGMEnt


```
<!--Element for Game Object-->
<gameObject id="fireman" type="actor">
   <attributes>
      <collidable value="true"/>
      <mass value="75.0"/>
      <inventory capacity="0.0"/>
      <appearance.value>
         <assetManager.load id="assetManager">
            <constant value="asset/fireman/sprite.png"/>
         </assetManager.load>
      </appearance.value>
      <animations>
         <spriteSequence id="idle">
            <frame top="0" bottom="20" left="0" right="20"/>
            <frame top="0" bottom="20" left="20" right="40"/>
            <frame top="0" bottom="20" left="40" right="60"/>
         </spriteSequence>
         <spriteSequence id="walkLeft">
            <frame top="0" bottom="20" left="60" right="80"/>
            <frame top="0" bottom="20" left="80" right="100"/>
            <frame top="0" bottom="20" left="100" right="120"/>
            <frame top="0" bottom="20" left="120" right="140"/>
            <frame top="0" bottom="20" left="140" right="160"/>
         </spriteSequence>
      </animations>
   </attributes>
   <operations>
      <gameObject.update>
         <if>
            <condition>
               <gameObject.actionState/>
               <operator value="=="/>
               <constant value="idle"/>
            </condition>
            <animationManager.setSequence>
               <gameObject id="fireman"/>
               <constant value="idle"/>
            </animationManager.setSequence>
         </if>
         <elseif>
            <condition>
               <gameObject.actionState/>
               <operator value="=="/>
               <constant value="walk"/>
            </condition>
            <physicManager.applyForce id="physicManager">
               <gameObject id="fireman"/>
               <constant value="0.5"/>
               <constant value="180"/>
            </physicManager.applyForce>
            <animationManager.setSequence>
               <gameObject id="fireman"/>
               <constant value="walkLeft"/>
            </animationManager.setSequence>
         </elseif>
         <animationManager.update id="animationManager">
            <gameObject id="fireman"/>
         </animationManager.update>
      </gameObject.update>
      <gameObject.render>
         <renderManager.render id="renderManager">
            <gameObject.appearance id="fireman"/>
         </renderManager.render>
      </gameObject.render>
      <gameObject.cleanup></gameObject.cleanup>
   </operations>
</gameObject>
```

Figure 16: XML definition of the game object in game technology model. elements in bold are translated token of information from game content model which has been reorganised into programmable format

Although we have gone through a lot of detail describing the technical aspects of the implementation it should be born in mind that the model driven engineering approach hides all of these from the actual domain experts and we include this only for completeness. From the model-driven serious

games development viewpoint, our Game Technology Model plays a significant role in the generation of interoperable software artefacts. In the context of GBL, our Game Technology Model (as well as Game Content Model and Game Software Model) and our model-driven serious game development framework is a point of reference for developers who wish to implement high-level authoring tools such as SeGMEnt for non-technical domain experts to produce a range of serious games for use in game-based learning.

8. Conclusion

In this paper, we have analysed the architecture of game engines, identifying the core components and presented our Game Technology Model based on our Game Content Model for use in our model driven approach in an attempt to address the lack of high-level authoring tools for serious games targeted at non-domain experts. Our Game Technology Model is a computational representation of game software that is independent of implementation platform and hardware platform. The introduction of the core components layer provides flexibility for specifying the preferred components and hence promoting a higher level independence over the choice of game technologies. This architecture may appear bureaucratic, but it is unlikely to raise many performance concerns for demanding real-time computation since models are used primarily to represent content and logic for use in the generation of software artefacts using MDE tools through different levels of refinement. Availability of a Game Technology Model and high-level authoring tools such as SeGMEnt provides the opportunity for domain experts to prototype-play-test-evaluate and make further improvement on the designs of serious games without having to face the technical barriers that exist in development of serious games for GBL.

References

Bézivin, J. (2004). In Search of a Basic Principle for Model Driven Engineering. *UPGRADE, V*(2), 21-24.

Bézivin, J., & Gerbé, O. (2001). *Towards a Precise Definition of the OMG/MDA Framework*. Paper presented at the 16th IEEE international conference on Automated software engineering, Coronado Island, San Diego, CA, USA.

Bishop, L., Eberly, D., Whitted, T., Finch, M., & Shantz, M. (1998). Designing a PC Game Engine. *IEEE Computer Graphics and Applications, 18*(1), 46-53. doi: 10.1109/38.637270

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2008, 5th December 2009). Extensible Markup Language (XML) 1.0 (Fifth Edition), from http://www.w3.org/TR/2008/REC-xml-20081126/

Carter, C., El Rhalibi, A., Merabti, M., & Price, M. (2009, 12-14 Oct. 2009). *Homura and Net-Homura: The creation and web-based deployment of cross-platform 3D games.* Paper presented at the Ultra Modern Telecommunications & Workshops, 2009. ICUMT '09. International Conference on.

Carter, C., Rhalibi, A. E., Merabti, M., & Bendiab, A. T. (2010). *Hybrid Client-Server, Peer-to-Peer Framework for MMOG*. Paper presented at the IEEE International Conference on Multimedia and Expo (ICME), Suntec City.

Darken, R., McDowell, P., & Johnson, E. (2005). Projects in VR: the Delta3D open source game engine. *Computer Graphics and Applications, IEEE, 25*(3), 10-12.

ECMA. (1997). Portable Common Tool Environment (PCTE) - C Programming Language Binding.

FAS. (2006). Harnessing the power of video games for learning, Summit on Educational Games 2006.

Favre, J.-M., & Nguyen, T. (2005). Towards a Megamodel to Model Software Evolution Through Transformations. *Electronic Notes in Theoretical Computer Science, 127*(3), 59-74.

Flatscher, R. G. (2002). Metamodeling in EIA/CDIF---meta-metamodel and metamodels. *ACM Trans. Model. Comput. Simul., 12*(4), 322-342. doi: http://doi.acm.org/10.1145/643120.643124

Gregory, J. (2009). *Game Engine Architecture*. Natick, Massachusetts: A K Peters, Ltd.

Järvinen, A. (2007). *Introducing Applied Ludology: Hands-on Methods for Game Studies*. Paper presented at the Digra 2007 Situated Play: International Conference of the Digital Games Research Association, Tokyo, Japan.

Jenkins, H., Klopfer, E., Squire, K., & Tan, P. (2003, October 2003). Entering The Education Arcade. *Computers in Entertainment (CIE), 1,* 17-17.

Kelly, S., & Tolvanen, J.-P. (2008). *Domain-Specific Modeling: Enabling Full Code Generation*. Hoboken, New Jersey: Wiley-IEEE Computer Society Press.

Kim, C., & Agrawala, A. K. (1989). Analysis of the Fork-Join Queue. *IEEE Trans. Comput., 38*(2), 250-255. doi: 10.1109/12.16501

Kleppe, A. G., Warmer, J. B., & Bast, W. (2003). *MDA Explained: The Model driven Architecture: practice and promises*: Addison-Wesley.

Ledeczi, A., Maroti, M., Bakay, A., Karsa, G., Garrett, J., Thomason, C., . . . Volgyesi, P. (2001). *The Generic Modeling Environment.* Paper presented at the Workshop on Intelligent Signal Processing, Budapest, Hungary.

Llopis, N. (2005). Game Architecture. In S. Rabin (Ed.), *Introduction to Game Development* (pp. 267-296). Hingham, Massachusetts: Charles River Media.

Sarinho, V., & Apolinário, A. (2008). *A Feature Model Proposal for Computer Games Design*. Paper presented at the VII Brazilian Symposium on Computer Games and Digital Entertainment, Belo Horizonte.

Schmidt, D. C. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer, 39*(2), 25-21.

Seidewitz, E. (2003). What models mean? *Software IEEE, 20*(5), 26-32.

Tang, S., & Hanneghan, M. (2010). *A Model-Driven Framework to Support Development of Serious Games for Game-based Learning*. Paper presented at the 3rd International Conference on Developments in e-Systems Engineering (DESE2010), London, UK.

Tang, S., & Hanneghan, M. (2011, 6-8 December). *Game Content Model: An Ontology for Documenting Serious Game Design.* Paper presented at the Proceedings of 4th International Conference on Developments in e-Systems Engineering (DESE2011), Dubai, UAE.

Tang, S., & Hanneghan, M. (2012). State-of-the-Art Model Driven Game Development: A Survey of Technological Solutions for Game-Based Learning. *Journal of Interactive Learning Research, 22*(4), 551-605.

Tang, S., Hanneghan, M., & El-Rhalibi, A. (2009). Introduction to Games-Based Learning. In T. M. Connolly, M. H. Stansfield & L. Boyle (Eds.), *Games-Based Learning Advancements for Multi-Sensory Human Computer Interfaces: Techniques and Effective Practices* (pp. 1-17). Hershey: Idea-Group Publishing.

Zagal, J. P., Mateas, M., Fernández-Vara, C., Hochhalter, B., & Lichti, N. (2005). *Towards an Ontological Language for Game Analysis.* Paper presented at the DiGRA 2005 - the Digital Games Research Association's 2nd International Conference, Selected Papers, Simon Fraser University, Burnaby, BC, Canada.