Tang, SOT and Hanneghan, M

State-of-the-Art Model Driven Game Development: A Survey of Technological Solutions for Game-Based Learning

State of the Art Model Driven Game Development: A Survey of Technological Solutions for Game-Based Learning

Stephen Tang* and Martin Hanneghan
Liverpool John Moores University, James Parsons Building, Byrom Street, Liverpool, L3 3AF, United Kingdom

* Corresponding author. Tel.: +44 151 231 2279
E-mail address: o.t.tang@ljmu.ac.uk (Stephen Tang).

Keywords
Architectures for educational technology system
Human-computer interface
Simulation
Interactive learning environments

Abstract
Game-based learning harnesses the advantages of computer games technology to create a fun, motivating and interactive virtual learning environment that promotes problem-based experiential learning. Such an approach is advocated by many commentators to provide an enhanced learning experience than those based on traditional didactic methods. However, the adoption of such a seductive learning method engenders a range of technical, educational and pedagogical challenges, including: (i) how to enable domain experts - with little computer games development skills – to plan, develop and update their teaching material without going through endless and laborious iterative cycles of software and content development and/or adaptation; (ii) how to choose the right mix of entertainment and game playing to deliver the required educational and pedagogical lesson/teaching material; and (iii) how to reuse existing games software frameworks and associated editing environments for game-based learning. Much research is already underway at addressing the stated challenges; however, these approaches do not address the key challenge of facilitating the planning and development of teaching material with the right mix of pedagogical elements, educational components and fun. Thus, this study aims to investigate the use of model-driven software engineering approaches to facilitate non-technical domain experts (teachers) to plan, develop and maintain game-based learning resources regardless of the intricacies of the game engine/environment (platform) used. This article investigates the state-of-the-art in model-driven game development to provide a summary of developments in game design languages, game software modelling languages, game models, game software models, model-driven game frameworks, game software frameworks, model-driven engineering tools and assistive user interfaces. The findings from this survey will prove a useful guide for future development of high-level educational game creation tools for game-based learning.

1. Introduction
Computer (video) games are interactive software applications created primarily for participatory entertainment purposes (Rollings & Adams, 2003). Today's application of computer games extend beyond entertainment to include health, advertisement, training, education, science, research, production and work, in which games technologies are used specifically for improving accessibility of simulations, modelling environments, visualisation, user interfaces, communication, learning and training, and productive activities such as authoring, development or production (Sawyer & Smith, 2008). In the recent years, studies on incorporating computer games into existing e-learning environments for use in institutions of higher learning have began to attract researchers around the world. Findings from initial research studies show that computer games can be used to acquire certain cognitive abilities and improve learners' understanding (Aguilera & Mendiz, 2003; BECTa, 2006; Jenkins, Klopfer, Squire, & Tan, 2003). These preliminary results are convincing and have gained tremendous interest from different sectors including government, academia and industry to further explore the benefits of this opportunity (BECTa, 2006; FAS, 2006).

Game-based learning refers to the innovative learning approach derived from the use of computer games that possess educational value or other kinds of software application that use games for learning and education purposes such as learning support, teaching enhancement, assessment and evaluation of learners (Tang, Hanneghan, & El-Rhalibi, 2009). The intent of game-based learning is to bridge the gap between learning and digital culture in the 21st century, making learning more relevant and appealing via persuasive technology to motivate learners to acquire knowledge and skills via a digital medium which is then transferable into reality. Game-based learning does not replace traditional learning, but can be offered as an alternative medium such as *Electronic Learning*, *Multimedia Learning* and *Mobile Learning* to address the changes in learning patterns among the 'PlayStation' generation of learners.

Positive findings from earlier game-based learning projects such as MIT's Games-to-Teach Project (Jenkins, et al., 2003) and BECTa's Computer Games in Education Project (BECTa, 2006) are paving the way for more domain experts to adopt this learning approach. However, there are great challenges ahead that require attention from stakeholders before learners can fully reap the benefits of game-based learning. One of the key challenges is the source of educational games to be used in a game-based learning setting. Sourcing educational games commercially is an attractive option but it is rather challenging to get the right content for use in an educational setting. Most commercial-off-the-shelf (COTS) games are designed specifically to entertain and a large number even elicit violence and sexual content, thus rendering them inappropriate (although this does not imply useless[1]) for use in an education context (Tang & Hanneghan, 2005). The other alternative then is to spearhead in-house development of game-based learning content using open source or royalty-free game engines and 'modding' (modifying) COTS games by utilising a game editor application to create customised game objects and levels. Almost all other options available require technical knowledge, which precludes most domain experts whose main aim is to solely impart their knowledge rather than become games programmers. It is evident that there are very few tools available to support development of game-based learning content for the novice or non-developer user group.

Development of computer games is both artistically and a technically demanding task. Game developers, who are usually creative individuals with a computing background, are the best candidates for the job. While game developers may appear to be the key to game-based learning they are available in such small numbers in relation to the vast number of domain experts who wish to produce educational games. In fact, many of the educational games development projects are collaborative efforts between domain experts and game developers (Jenkins, et al., 2003; H. Kelly, et al., 2007). However, professional expertise does not come cheap. Game developers typically rely on a game software framework (game engine) and other

[1] Controversial COTS games such as Grand Theft Auto have been used by the police force to educate and deter primary school children in Merseyside, UK from aggressive behaviour (http://news.bbc.co.uk/1/hi/education/8611536.stm). This demonstrates that COTS games can still be used for game-based learning. Domain experts will have to be selective in choosing the right COTS games for use in game-based learning to ensure effective learning take places.

custom tools to produce computer games quickly, affordably and reliably. Most of these tools have a very steep learning curve and may require a substantial amount of technical knowledge about game development. Such requirements are huge challenges to most non-technical domain experts who have little knowledge about game development. Nevertheless, similar tools can be developed specifically to guide and facilitate a programming-free game development environment for the non-developer user group.

In terms of engineering educational games, the underlying technology and development methods remain the same as mainstream entertainment games. The only notable distinctions are in the definitions of *rules*, *play* and *aesthetic representation* (usually defined by game designers but in the context of educational games this is the role of the domain expert). Such characteristics allow educational games to be represented in a formal design specification. In a Model-Driven Engineering (MDE) approach, this formal design specification (excluding art and technical aspects) is represented in the form of models. A domain expert can express educational games through modelling. These models can then be analysed and transformed into code for a targeted game framework. This novel approach opens up the door for non-developers to create computer games affordably, reliably and quickly. In an attempt to help realise game-based learning, our research led us to query what the present advancements are in the field of game design, game engineering and MDE which can direct research on model-driven game development in the future.

In this article, we briefly describe MDE and explain the benefits of this approach in aiding domain experts to create resources suitable for game-based learning – the source of educational games or serious games. The body of this article presents the existing approaches to represent computer games ontologically both from game design and game software perspectives. In addition, it studies the characteristics of existing game design techniques and software modelling techniques for modelling computer games. It also surveys a selection of game software frameworks and model-driven game frameworks. Finally, it discusses a collection of model-driven engineering tools and assistive user interfaces appropriate for an educational game modelling environment. These studies provide new answers to how we can capitalise on existing technologies to drive the development of a model-driven framework for educational games forward and take a leap on the realisation of mass adoption of game-based learning.

2. Model-Driven Engineering (MDE)

MDE refers to a software development approach that relies extensively on the use of 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 the models. This approach can help in alleviating the platform complexity and expressing domain concepts effectively (Schmidt, 2006). Another term synonymous to MDE is Model Driven Development (MDD). In this article, MDE and MDD are used interchangeably.

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" and that the model "…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 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). Alternatively, it can also be used as a specification for beginners to study a system. 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.

MDE is domain specific, 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 (S. Kelly & Tolvanen, 2008). Models are then translated from source model to target model throughout the pipeline of MDE and finally to a complete implementation of a software system (or documentation of a complete software system). The relationship between the source model and the target model is termed as model transformation (France & Rumpe, 2007). The transformation process is a unidirectional process (S. Kelly & Tolvanen, 2008) which can be done either manually or automated using MDE tools.

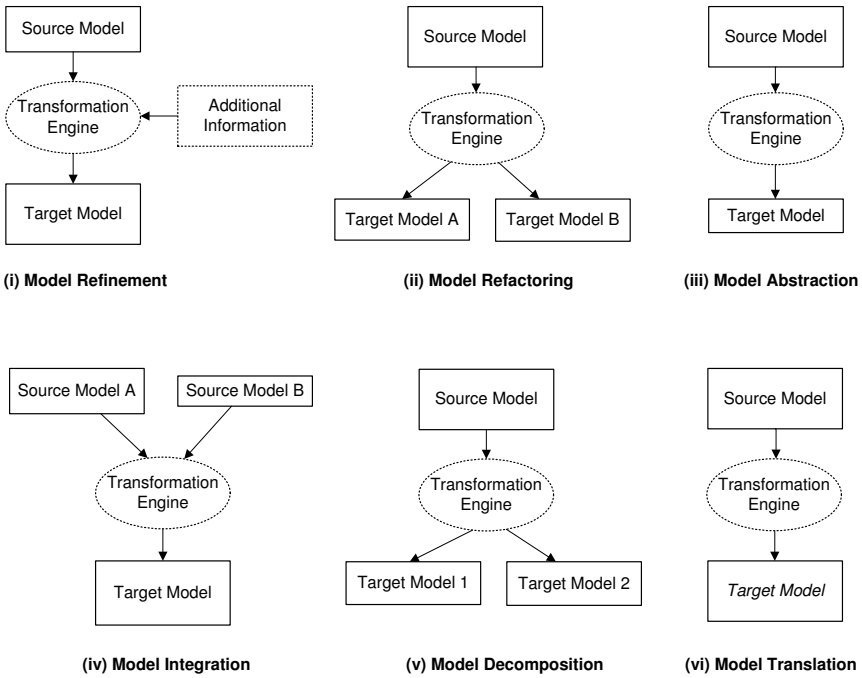

Figure 1: Model-to-Model Transformations

Transformation can either be a *model-to-model* transformation or *model-to-artefact* transformation. In a *model-to-model* transformation, the form of transformation includes: (i) *model refinement* where a source model is added with additional information to form a target model; (ii) *model refactoring* where a source model is producing multiple target models with different viewpoints; (iii) *model abstraction* where a source model has certain information discarded to produce a target model with higher abstraction; (iv) *model integration* where source models with different viewpoints are integrated to create a target model with combined viewpoint; (v) *model decomposition* where a single model is decomposed into multiple target models; and (vi) *model translation* where the source model is expressed in different languages (France & Rumpe, 2007). Such processes can be automated using MDE tools generically known as *transformation engines*. As for the *model-to-artefact* transformation, transformation can be in form of *code generation* where the model is translated directly into a software artefact such as scripts, configurations, codes or middleware languages which is accepted by the targeted platform, or *documentation generation* where specifications are represented in the form of textual or graphical output such as UML diagrams. Generators are MDE tools that are suitable for such transformations.

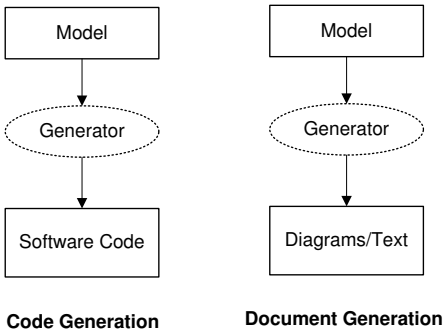

Figure 2: Model-to-Artefact Transformations

The number of model-to-model transformations required for an end-to-end transformation depends on the level of abstraction in the model driven framework. One-tier architectures can directly transform a model into code, but multiple-tier architecture such as the Object Management Group's (OMG) Model Driven Architecture (MDA) would have to undergo two cycles of model-to-model transformations before it can perform model-to-artefact transformation.

The MDE process can be represented with two distinct approaches; *top-down* and *bottom-up*. Both approaches are driven by different motivations. Justified by the need of generating a variation of software code, the top-down approach examines variations from the conceptual level. This approach can result in the creation of better designed, robust and consistent models. Defining a complex model may seem straightforward and easy but implementing such a model can be daunting (Cook, Jones, Kent, & Wills, 2007). The top-down approach is defined as following a succession of activities (Fondement & Silaghi, 2004):

i. Identifying the level of abstraction and platforms to be integrated;
ii. Specifying modelling notation and abstract syntax to be used at each level of abstraction;
iii. Specifying refinement processes, platform and related information to be integrated in lower level of abstraction;
iv. Defining generators for a modelling language used at the lowest level of abstractions (and even deployment of such code);
v. Specifying a verifier and validator to check against the upper level model, and generation of test cases for the system under development.

Contrary, the bottom-up approach addresses software variability through observing variation in existing codes. It is a client-oriented approach that requires more effort, consumes time, and may limit the extension of models in the future, however it is simpler to implement. The bottom-up MDE process is defined by Cook, Jones, Kent, & Wills (2007) as follows;

i. Identifying variability from business concepts;
ii. Conducting variability analysis on the existing codes;
iii. Defining a domain model from analysis and business concepts;
iv. Specifying modelling notation for the DSML;
v. Refining the DSML;
vi. Defining a generator based on existing code;

Cook, Jones, Kent, & Wills (2007) advise that it is best to alternate between the two approaches and work incrementally to ensure models and transformations are consistent.

Over recent years researchers have strived to improve MDE technologies applying lessons learnt from earlier efforts in developing platforms and languages with high-levels of abstraction. Lately researchers have opted to adapt the MDE approach deviating away from traditional software development approaches with the intention to separate the model from a platform dependent implementation. MDE approaches have been applied in areas such as web application modelling (Schauerhuber, Wimmer, & Kapsammer, 2006), web services modelling (Grønmo, Skogan, Solheim, & Oldevik, 2004), security modelling for distributed systems (Lodderstedt, Basin, & Doser, 2002), QoS validation of component-based software systems (Hill, Tambe, & Gokhale, 2007), control and automation systems modelling (Thramboulidis, 2004), user interface modelling (Sottet, et al., 2006), business software modelling (Hildenbrand & Korthaus, 2004), content repurposing (Obrenovic, Starcevic, & Selic, 2004) and context-awareness web application modelling (Ceri, Daniel, Facca, & Matera, 2007). Current research in MDE focuses mostly on systematic reuse of development experience through the concept of a software factory, systematic software testing and compilation technologies (France & Rumpe, 2007). There is now an abundance of knowledge and experience about MDE in the scientific community to push for industrialisation of educational games to accelerate the wide adoption of game-based learning.

2.2. Advantages and Disadvantages 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 to support software maintenance (Kleppe, Warmer, & Bast, 2003). In addition, MDE can also lead to better quality applications due to the integration of domain rules into the modelling language which minimises modelling error and promotes reliable mapping from model to code (S. Kelly & Tolvanen, 2008). For an obvious advantage, MDE encapsulates the technical aspects of development and therefore provides the necessary aid to domain users to develop applications with minimal low-level technical

knowledge. In addition, process, best practices and pedagogic guidelines for designing educational games described in our earlier work (see (Tang & Hanneghan, 2010)) can be embedded within the MDE environment to guide domain experts to produce educational games whilst assuring the quality of educational games design.

One major downside to MDE is the high cost of development which may or may not be justifiable with the amount of turnover a project can generate. Overall, MDE can offer great benefits for businesses and for non-technical domain users as it opens the door to create quality, reliable applications easily and affordably.

2.3. Challenges

The benefits of MDE are certainly attractive to many, but realisation of an MDE solution is still challenging for developers. France & Rumpe (2007) classified the challenges of MDE into three categories as follows:

- *Modelling Language Challenges*: When defining a modelling language, language developers are challenged to create the problem-level abstractions and formalise the semantics of the language. Creating the problem-level abstractions requires the language developer to fully understand the concepts, terminologies and relationships that represent the problem domain. This task can be made easier with the help of domain experts. While creating a notation for the language, iteration of user testing will help to ensure the notation is acceptable and usable.
- *Separation of Concerns Challenges*: In complex software, problems may arise when modelling involves multiple and overlapping views using heterogeneous language. Some features of the system may have conflicts due to the design and it is a real challenge for developers to make sure that features modelled can interact with each other. Separation of these design concerns is necessary to avoid costly system failure and can be achieved through a well-defined viewpoint.
- *Model Manipulation and Management Concerns*: More challenges await developers in matters pertaining to transformation of models. These include maintaining the consistency of models transformed, testing of transformation and integration of generated code with foreign code. The transformation process also adds complexity to management of models when revisions are made to models in terms of versioning, repository storage of previous models and maintaining integrity of relationships of models.

These challenges are greater for complex software such as educational games. In addition to functional aspects of educational games, the MDE developers will have to address the incorporation of media such as 2D graphics, 3D models, sounds and animation data, and the extensibility of game functionalities such as game physics and game artificial intelligence (AI) while assuring educational games produced are of acceptable quality. The MDE approach may also limit innovation of educational games due to the constraints of customisation. Although this is a trade-off for the MDE approach, it is of less concern for domain experts as the benefits listed in Section 2.2 far outweigh it.

3. State-of-the-Art Model Driven Game Development

Game-based learning has some of the most desirable learning approaches embedded such as active learning, experiential learning, situated learning and problem-based learning (Tang, et al., 2009). There are a growing number of domain experts keen in adopting this learning approach and the numbers will continue to grow with more realising the potential that computer games hold (Futurelab, 2006). Learners themselves are ready for such approaches as they are increasingly familiar with gaming culture (Pearce, 2006) and the technology to support game-based learning is more affordable and widely available. The only puzzle to realise the vision of game-based learning is the availability of high-level tools made specifically for non-technical domain experts to assist them in creating educational games in an affordable and timely manner. As highlighted earlier in this paper, MDE offers the basis for development of high-level tools. Architecting a model-driven educational games development environment will require an educational game modelling language, a framework that can integrate media and game functionalities, transformation engines and code generators. In this section, we survey the literature in the domain of game design, game software engineering and MDE, and summarise the key developments that are propelling games development towards the model driven approach.

3.1. Game Design Languages

Design is generally defined as “the human capacity to shape and make our environment in ways without precedent in nature, to serve our needs and give meaning to our lives” (Heskett, 2005). From a game design perspective, it is regarded as the creative expression of a game designer’s vision of some virtual goal-directed play where the game designer engages him or herself in an activity to innovate and create a concept of play within a set of business, psychological, sociological and technological constraints. This activity demands documentation to record the decisions on choices which serve as a blueprint to communicate the underlying vision to the development team.

Conventionally, game design is documented as a collection of game design documents in the form of high-concept document, treatment document and detailed design document. This approach is still taught through texts (see for example Rouse (2001), Rollings and Adams (2006; 2003), Bates (2004) and Fullerton (2008)). Documenting the aspects of game design is a somewhat informal process driven by creativity (Crawford, 1982; Fullerton, 2008). Design within a specific game genre is also bound by rules and certain expectations. For example, when designing a role-playing game (RPG) such as the Final Fantasy[2] series, emphasis should be given on design of a customisable player-character that can be improved throughout game-play, a story built around a quest that can engage game players in an explorative world to satisfy the appetite for adventure. In contrast to this, construction and management games such as the SimCity[3] series have their design focused on the economy system and the construction process. At present, samples of game design documents and templates for game design from game design references (Fullerton, 2008; Rollings & Adams, 2003; Rouse, 2001) are the only available guide for documenting game design.

In recent years, various studies (Adams, 2004; El-Rhalibi, Hanneghan, Tang, & England, 2005; Onder, 2002; Tang, Hanneghan, & El-Rhalibi, 2004; Taylor, Baskett, Hughes, & Wade, 2007; Taylor, Gresty, & Baskett, 2006) have proposed the use of modelling languages to help game designers systemically and systematically design computer games. Our definition of an ideal game design language would consist of game-related vocabularies and game-concept related grammars that allow game designers to formally express their game idea. The *Soft Systems Methodology* (SSM) (Checkland & Scholes, 1990), an approach based on the theory of systemic thinking to study the components of a system and their relationships with one another in order to develop an understanding about that system. This could be used as a formal method for designing games. In the context of games design, SSM can assist game designers in designing game flow and identifying appropriate challenges and conflicts (El-Rhalibi, et al., 2005; Tang, et al., 2004; Taylor, et al., 2007). A game idea is elaborated in the form of a *Rich Picture*™ where purposeful activity systems are identified and a relevant *Root Definition* (RD) is defined based on *Customers, Actors, Transformation process, Weltanschauung (or world view), Owners* and *Environmental constraint* (CATWOE) analysis. An example of this is illustrated in Figure 3. SSM offers a methodological and systemic approach to game design through an iterative process of logical reasoning, but it lacks aspects that describe the story, user interfaces, input control, objectives and game-play which are essential for creating compelling educational games.

[2] Read more about Final Fantasy from http://www.playonline.com/ff11us/index.shtml.
[3] More details about SimCity can be found at http://simcitysocieties.ea.com/index.php.

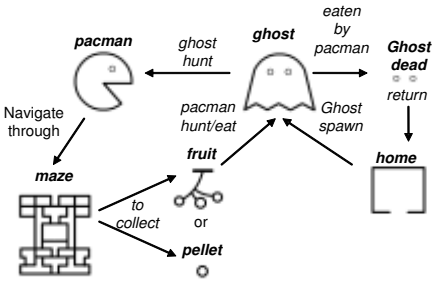

Figure 3: Pacman Game represented in Rich Pictures (Tang, et al., 2004)

The *story beat diagram* proposed by Onder (2002) is composed of ovals and arrows that represents progression of a story (see Figure 4). It illustrates the possible routes for the game player to explore within the defined interactive story. In addition to the story beat diagram, further additional details on objects and casts members, locations of scenes, setup of scenes, player interactions and associated responses, and distribution of game-play within the storyline are required in writing to complete the specification of a game design. Essentially, the story beat diagram can only capture the flow of game-play ordered within a story while the rest of the requirements are recorded separately.

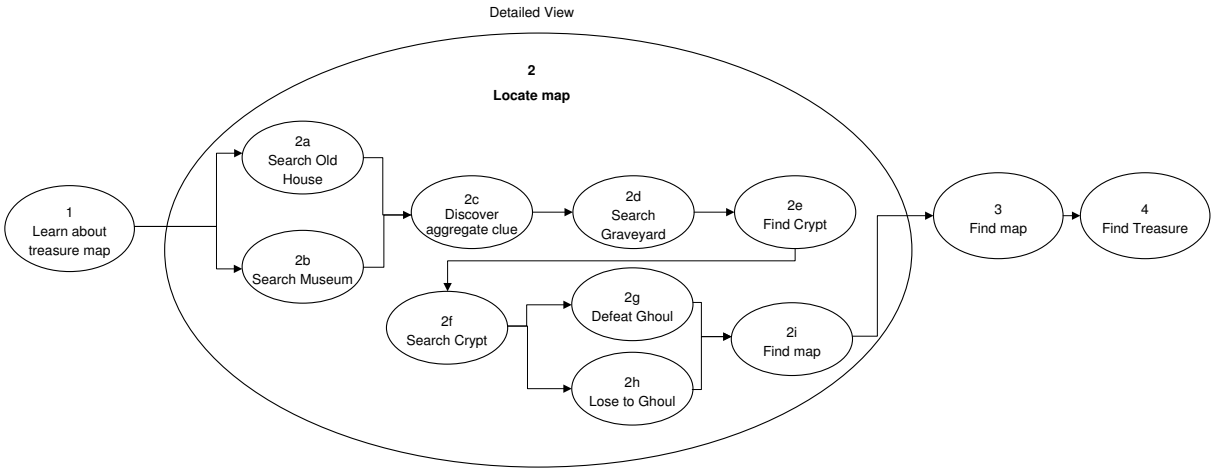

Figure 4: Example of Story Beat Diagram (Onder, 2002)

The *flowboard* is an adaptation of the storyboard and flowchart to document game structure (Adams, 2004). It uses rectangles to denote segments of a game such as splash screen, menu screen, result screen, game screen, high-score tables and other, while arrows with conditions denote the constraints or condition that allow progression from one segment to another (see example in Figure 5). The flowboard is a simple and useful tool to model the various modes offered in a game, but it lacks the ability to record essential game design details required to complete the design of a game such as objects and character's behaviour, challenges, conflicts, objectives, environment, user interfaces, storytelling and narrative.

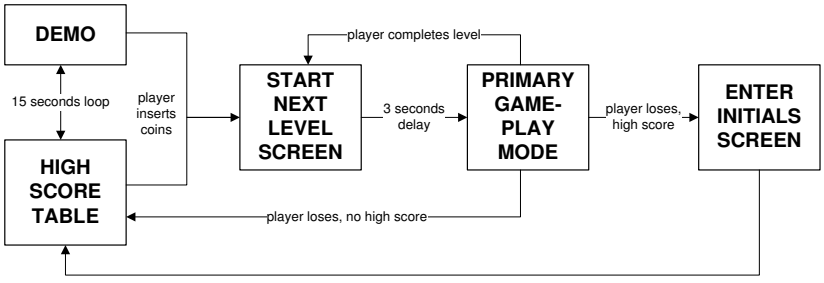

Figure 5:Example of Flowboard (Adams, 2004)

There have also been efforts to use software modelling languages such as *the Unified Modelling Language (UML)* for game design. The *Computer Game Flow Design Diagram* proposed by Taylor, Gresty, & Baskett (2006) adapts the language from the UML use-case diagram to form a new diagramming language to document game flow. The computer game flow diagram uses arrows to denote the direction of game-flow, rectangles for fixed game objects, ovals for mobile game objects (game objects that move around) and pseudo code to describe interactions and responses (see example in Figure 6). In addition to modelling linear game flow, it can also be used to model non-linear flow where games can be designed to provide game players greater freedom to navigate within the game world achieving game objectives in a non-ordered fashion. The use of symbols and labelling of objects, characters and environments are clear, but the method is insufficient for describing details of objects, characters, objectives, environments,

narrative and storyline. Although the use of pseudo code can help to describe many more details programmatically, this approach is too technically demanding for non-technical domain experts to document design details adequately.

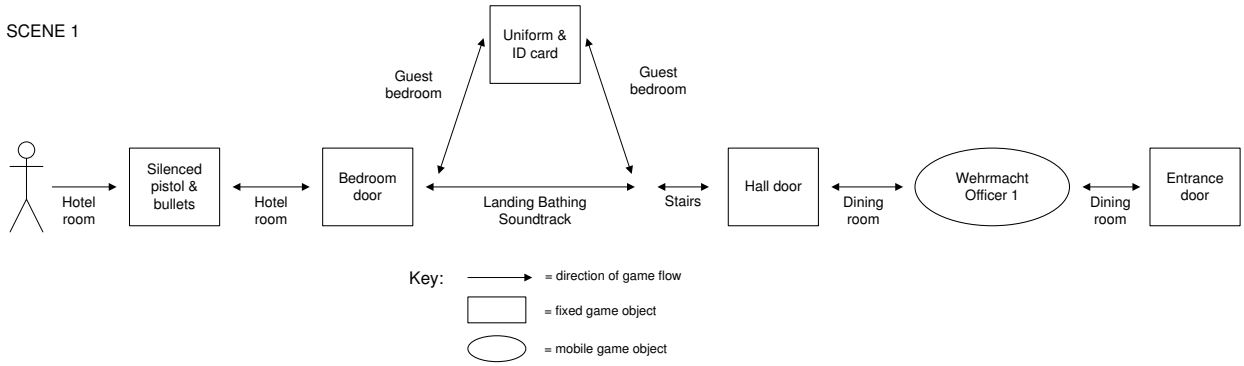

Figure 6: "On track" level of 'Medal of Honour: Frontline' represented with UML Use-case Diagram (Taylor, et al., 2006)

Natkin, Vega, & Grünvogel's (2004) approach to modelling game design addresses the need to model spatial-temporal relationships through the adaptation of *Petri-nets* and the use of generalised graphs called *hypergraphs*. They use special Petri-nets composed of three relations (*transaction a and transaction b is not related*, *transaction a is before transaction b*, and *if transaction a is executed then transaction b cannot be executed*) between two different transactions (actions of the player) to represent the temporal relationships (order of game sequence) in a game and game levels. Sets of transactions are then translated into a hypergraph to define the spatial relationship (events in relation to the map in the game world). In this approach, a circle is used to denote a place, a square or rectangle for transactions and arrows denote an input arc (which connects a place with a transaction) and output arc (connect transaction with place) (see Figure 7). Using this approach, game designers can model flow and events for game and game level. However, it cannot be use to document details on objects, characters and their behaviour, environment, narrative or storytelling.

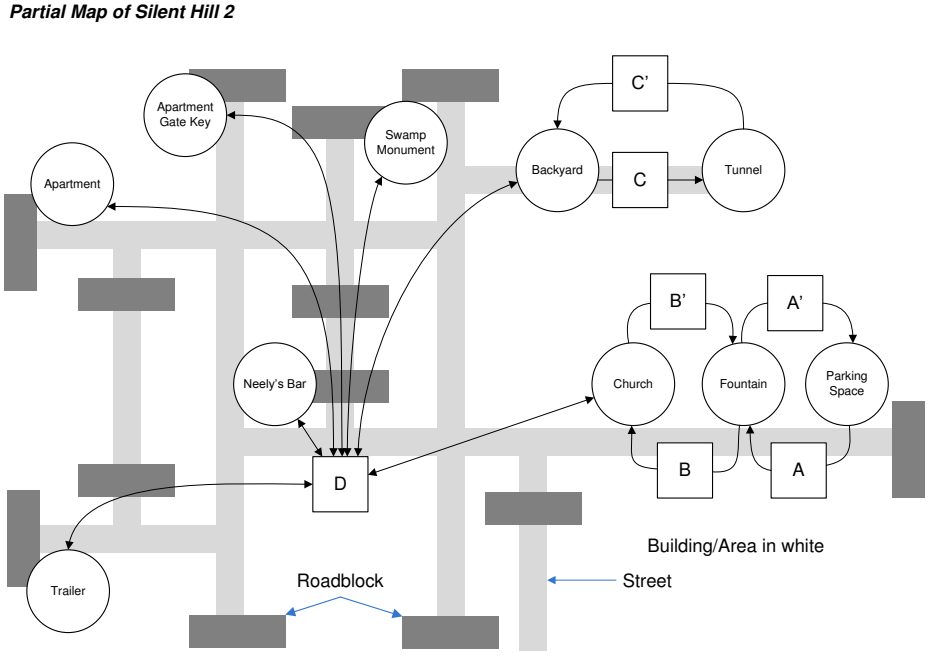

Figure 7: Partial Map of Silent Hill 2 represented in Hypergraph (Natkin, et al., 2004)

The formal approaches of documenting game design reviewed only focus on documenting game flow formally using various graphical modelling languages, while other aspects of the game design remain informal and subjective for others to interpret using text description. Although these approaches help games designers in one or more ways, there are still many aspects of game design which can be documented formally which have yet to be studied while others such as visual, audio, storytelling cannot be easily formalised and will continue to be documented using current approaches i.e. textual descriptions or sketches. It is highly desirable to formalise documentation of game design as it can enhance the effectiveness of translation of the design to software code.

3.2. Game Software Modelling Languages

Modelling is a formal process of documenting a design using syntactically composed reserved words or graphical notations. Similarly, the term modelling can be used for the approaches described in Section 3.1, but it refers to documenting aspects of game design. Through modelling, a conventional software engineer captures the necessary requirements to produce a concrete model that best represents the software specifications of

the game software. In addition, modelling allows the engineer to study problems and look for flaws in the design prior to development of the software system. Flowcharts, Data Flow Diagrams (DFD), Statecharts, Z-Notation, Petri-nets, Business Process Modelling Notation (BPMN), Integrated Computer-Aided Manufacturing Definition Language (IDEF) and UML are some examples of software modelling languages that engineers can use to express the design of the game software.

Modelling was historically mathematically demanding and modelling languages were developed to suit programming languages during the era such as FORTRAN, ALGOL, LISP and COBOL. As later generation programming languages such as C, Pascal and Ada which were designed to structure programs logically began to surface, software modelling took on a function-oriented paradigm. At present, Object-Oriented (OO) modelling is a de facto approach in the software industry as it promotes abstraction, encapsulation and reusability of software code.

Although most software modelling languages mentioned can be used to model aspects of game software, only a few are actually used in practice. *Tokenization* (Rollings & Morris, 2004) represents each game object (both interactable and non-interactable) as tokens. Interactions between tokens are represented in a triangular matrix as shown in Figure 7. Tokenization invites game developers to think in tokens and the associated interaction, but lacks many aspects necessary to represent the game software in its entirety. Simple casual games can be modelled using this approach without much complication. However, for modern computer games that offer more elaborate interactions and a greater number of tokens, this approach is not suitable as complexity of modelling increases exponentially. Another major setback for tokenization is evident during translation of model to software code. Although examples are available, it is not a straight forward process as there is much information such as the game object's behaviour, setup of scenarios, flow of events and progressions still missing in the software model.

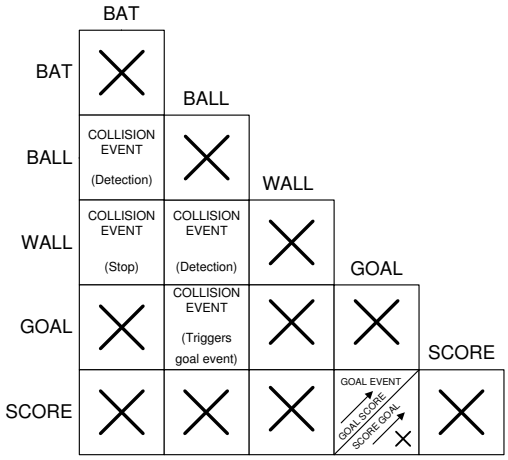

Figure 8: Tokenization Interaction Diagram of Pac Man Game (Rollings & Morris, 2004)

As many aspects of games deal with a collection of states that change based on interaction and rules, *Statecharts* or *Finite State Machine* (FSM) diagrams are ideal candidates for modelling games. The *FSM* has been mostly used in modelling game AI (Kienzle, Denault, & Vangheluwe, 2007), although other game aspects such as character behaviour, game flow and game scenario can also be modelled with this approach. Rollings & Morris's (2004) approach is a pictorial version of FSM with slight modifications in notation for modelling a game object's behaviour (see Figure 8). A square is used to represent an individual state of a game object, while a circle denotes an event. Lines with a coloured dot at the end show the transition between states and the incoming event that triggers a transition. Coloured dots are used to pair the event to the transition between states in cases where there are more than one event referring to the same state. A circular arrow is introduced in the FSM diagram to include events that can cause entry to a state. Kienzle, Denault & Vangheluwe's (2007) approach uses a variant of *Rhapsody Statecharts* (Harel & Kugler, 2001) which combines statecharts and class diagrams to represent properties and behaviour of game objects (see Figure 9). In addition to class diagrams, statecharts are accompanied to switch or alter the state of a game object when certain conditions are met. Rounded rectangles represent the state while transition of state is shown using an arrow with conditions. Statecharts can be nested to create a multi-level statechart, but every statechart can only have a single begin state which is denoted by a filled black circle. Rhapsody Statecharts are a better approach than the pictorial FSM or classical statecharts as they represent both properties and methods of a software component and can easily used for representing other aspects of game software.

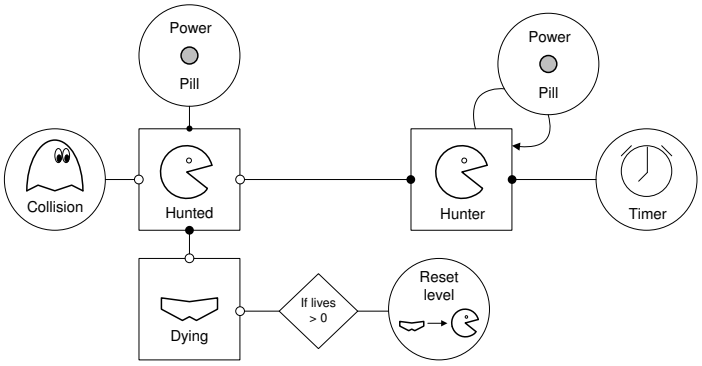

Figure 9: Statechart Diagram for Pacman game (Rollings & Morris, 2004)

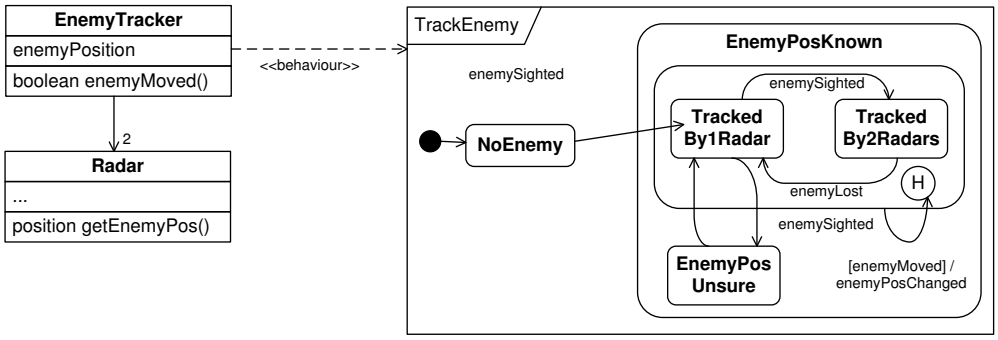

Figure 10: Rhapsody Statecharts of EnemyTracker AI component (Kienzle, et al., 2007)

As highlighted earlier, OO is the preferred approach and most computer games software is modelled using OO with the industry standard software diagramming language UML (Gal, Prado, Natkin, & Vega., 2002). This approach makes it simple for game developers to translate game software requirements to OO programming languages. UML consists of thirteen types of diagram to document three aspects of software systems: *functional*, *structural* and *behavioural*. There have been several proposals for modelling games software using UML. Ang & Rao (2004) represent player interaction with tokens using UML use case diagrams (see Figure 10) in an attempt to identify all possible interactions in the game before representing it in a class diagram. The identified interactions are behaviours of relevant classes in the game and it helps to discover other necessary classes to complete the software representation of the game. Bethke (2003) illustrates how a range of UML diagrams can effectively document design of a game software. For example, the UML use case diagram in Figure 11 is used to represent a player's interaction, while the UML class diagram featured in Figure 12 is used to represent game objects and static structure. Reyno & Cubel (2008) use UML class diagrams and state transition diagrams to represent structure and behaviour of game (see Figure 13). UML class diagrams can be used to represent the association between game objects represented as classes which can also be represented using *object diagrams* to examine the run-time static relationship behaviour. A collection of classes can be ordered into a high-level view using a component diagram, while a deployment diagram can be used for representing the physical arrangement of the game possibly utilising an external game engine. Activity diagrams can be an alternative to statecharts for modelling behaviour of game objects. In addition, sequence diagrams can be used to extend modelling of behaviour between game objects. UML certainly offers a range of useful diagramming approaches to document games software. The choice of UML diagram to use depends on how such diagramming tools help game developers to visualise the game software under construction and translate the specification to software code.

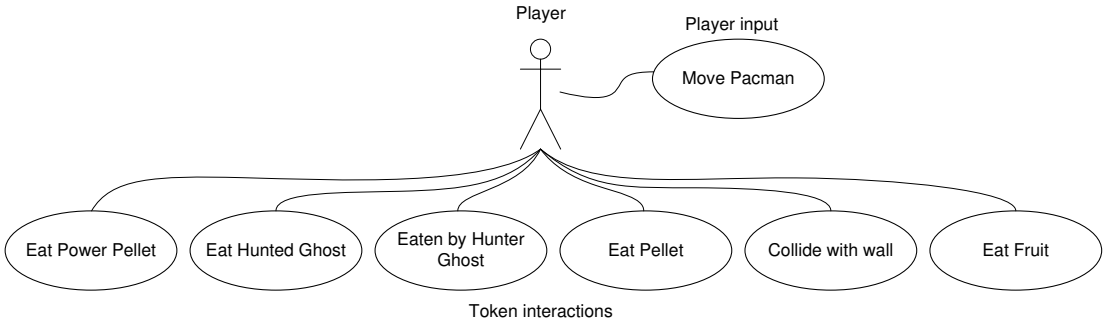

Figure 11: Use Case Diagram of Pacman Game (Ang & Rao, 2004)

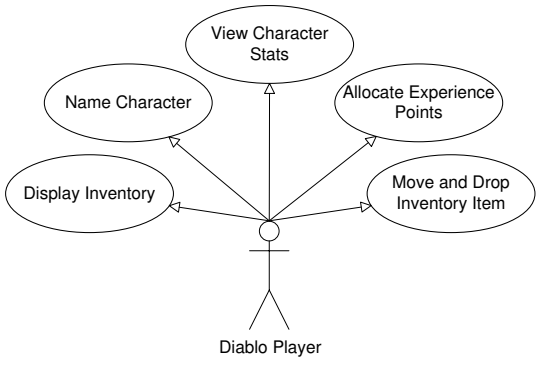

Figure 12: The character transaction use cases of Diablo (Bethke, 2003)

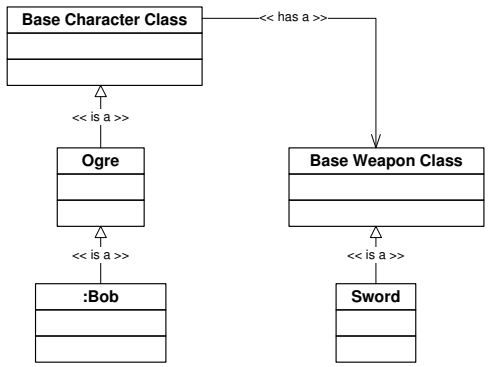

Figure 13: Example of Class Diagram (Bethke, 2003)

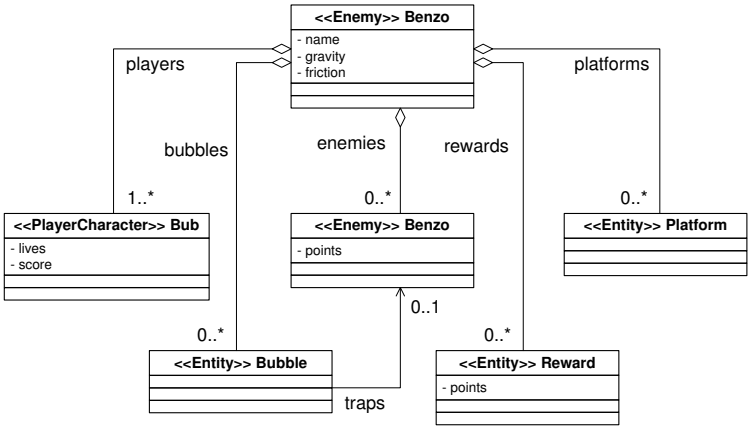

Figure 14: Structure Diagram of Bubble Bobble (Reyno & Cubel, 2008)

The reviewed game software modelling languages do not represent the entire collection of techniques for modelling game software. Many other techniques exist to model games software such as Flowcharts, BPMN, DFD and IDEF. In practice, a combination of software modelling techniques can be used to model the different aspects of game software as it is not always possible to represent all aspects of games software to be modelled using a single modelling language or notation.

3.3. Game Meta-Models

Game meta-models are templates which game designers can use to complete the design of a game by providing the missing details required. It encases game technologies with purpose, rules, play and aesthetic representations (and pedagogy as well in the context of educational games) as creative and instructional expression of a knowledge domain. Game meta-models can be used with the aforementioned game design languages in Section 3.1 to guide amateurs and novices in producing game models that conform to the game meta-model. However, the completeness of a design specification is still very dependent on the attention to detail of the individual game designer. A game model provides the platform for game designers to complete the design of a game by simply providing the necessary details required by the game model.

In recent years, there are a growing number of formal approaches developed to aid game designers to analyse, describe and study computer games. Hunicke, LeBlanc, & Zubek's (2004) Mechanics, Dynamics and Aesthetics (MDA) framework provides a guide for game designers to specify a game design systematically by focusing on a definition of aesthetic using a taxonomy for fun, the underlying system which supports the aesthetic requirements and finally the actions, behaviour and control mechanisms made available to game players. The taxonomy of fun consist of vocabularies such as *sensation*, *fantasy*, *narrative*, *challenge*, *fellowship*, *discovery*, *expression* and *submission* which, when used to describe play experiences, can help game designers think about the underlying dynamics and mechanics of a game. It is an interesting approach to design games in a structured manner while still fostering creativity. However, it only focuses on the aspects relating to game-play and has no solid structure to formally represent the components that make up a game such as game objects, characters, environment, user interfaces, storytelling, narrative and game progression.

Björk, Lundgren & Holopainen (2003) approach game design from a research-driven perspective by cataloguing descriptions of recurring interactions that relate to game-play which they term as *game design patterns* for analysing and studying games. Each game design pattern has a name, core definition of the pattern, a general description, usage description of the pattern, a description on the consequences of this game design pattern, a description of the relationships between other game design patterns and references that describe the origin of the pattern. To date there are more than 200 patterns identified and these are organised into collections of *game elements; resource and resource management; information, management and presentation; action and events; narrative structures, predictability and immersion; social interaction; goals; goal structure; game session; game mastery and balancing;* and *meta game, replayability and learning curves* (Björk & Holopainen, 2004). The use of patterns and templates to document aspects of game design is a formal attempt and it promotes better organisation and structure in terms of documenting details on game play compared to MDA for example. Although the original intention of this approach is for analysing and studying existing games, the collection of game design patterns can now be used as a reference and model for defining components of future games.

The Game Ontology Project (GOP) (Zagal, Mateas, Fernández-Vara, Hochhalter, & Lichti, 2005) addresses the need of formal game design through the identification of both common and distinct elements of game design abstractly, and orders these hierarchically to form a game ontology. The game ontology is regarded as a language for game design and also a framework for game designers to improve their understanding on what to design in computer games. At the highest-level, the ontology focuses on the definition of *interfaces* that map user inputs to a set of allowable actions of an avatar; *rules* that govern the interactions and economies in the game world; *goals* that determine a player's success or failure in the game; *entities* that populate the game world; and *entity manipulations* which define how entities behave physically in the game world. Each ontology entry is labelled with a name, listed with relationships with other elements in the hierarchy and given a description about the element as well as examples of both concrete and partial reification of the elements. Game designers can make references to the ontology and design games by filling the descriptions of the game design. Formalising the descriptions of the ontology entries are still necessary to ensure such a game model can be translated to software code.

Rapid Analysis Method (RAM) (Järvinen, 2007) is a set of methods for analysis and design of computer games from a system viewpoint and players' viewpoint focusing on the emotional and the socio-physiological aspects of game-play. It consists of seven tools to aid amateur game designers to systematically identify and analyse game elements, game mechanics and the goal they are related to, permissible interactions, eliciting conditions for emotions during game-play, game rhetoric and to study the anatomy of a game. RAM is a comprehensive approach for studying games and is suitable for use as a game meta-model. Nevertheless, there are still missing details required for describing computer games in terms of attributes of game elements instead of building a taxonomy for cataloguing common game elements.

Fullerton's (2008) model represents a computer game from a closed-formal system context by studying the formal elements and dramatic elements that compose a game. *Objectives*, *procedures*, *rules*, *resources*, *conflict*, *boundary* and *outcome* are the formal elements that define the structure of a game, whereas *play*, *challenge*, *premise*, *character* and *story* are the dramatic elements used to encase the formal elements in order to engage the game players. This is a rather comprehensive representation of computer games that encompasses both technical and artistic aspects of game design, but yet still incomplete. There is one dramatic element missing: *game objects* that fill the game world. It may have most of the elements to represent a game but it lacks the formalised language necessary to describe these elements accurately.

Nelson & Mateas' (2007) model focuses on four domains; *abstract game mechanics* which regulate the operational aspects of a computer game, *concrete game representation* which presents the abstract game mechanics through audio-visual means, *thematic content* which are graphics and sounds created to represent the game world, and *control mapping* which relates the player's input to methods that modify abstract game state. The

simplicity of the model helps one to understand the computer game as a system easily, but it is a rather brief model to represent complete games as many aspects of game design remain widely undefined making it a less favourable candidate for a game meta-model.

The Narrative, Entertainment, Simulation and Interaction (NESI) model proposed by Sarinho & Apolinário (2008) is a game feature model. It organises four main features of game design into a hierarchical order: *narrative* which consists of flow that progresses the game with different goals governed by rules that determines the success or failure of player; *entertainment* which is made up of a theme for the game and the player's immersive experience during the game play; *simulation* which is composed of game elements such as characters, items and buildings that populate the game world and the relationships among them; and finally, *interaction* that maps the game player's control to the game-play features and the presentation of the game through audio, visual or haptic devices. The NESI model is another possible candidate for a game meta-model.

The game meta-models reviewed provide interesting ways to document and analyse game design. Each of these has a different focus on aspects of game design and approach to documenting the details of game design. Among all the approaches reviewed, the MDA is probably the least qualified candidate as a game meta-model due to the lack of structure and identification of elements that represent games in their entirety. Nevertheless, it is a great tool that can aid the game designer to think creatively when designing fun and engaging game-play. Contrary to that, the game design patterns approach is a good candidate for a game meta-model. Game design patterns provide a template for documenting the multiple aspects of game design and organises these into collections which can be used as references for novices and amateur game designers to create their own games. It, however, lacks formalised descriptions of game design patterns required in MDE. The GOP has a more logical organisation of the elements of games compared to game design patterns and demands relational information which is crucial for writing software code. Similarly, it lacks formalisation in describing each ontology entry. The RAM has a good method to identify, analyse and study about game elements, but it also lacks a method to formally describe these elements. Nelson & Mateas' model promotes simplicity in describing a game but lacks refined structure, categories in design details and a formalised method of describing the game which also demotes itself as a candidate for game meta-model. The NESI model is a useful approach to describe games through description of features, but it fall short on how one can formally describe these features. By 'formal' we mean the use of a single unified language that may be human or machine-readable. Fullerton's model is by far the most comprehensive model reviewed, but it still lacks one element (i.e. game objects) and has no clear methods to document these elements except from traditional form of creative writing. It is clear that the one common shortcoming these approaches share is the lack of formality and details to accurately represent a computer game. Such an issue must be addressed to ensure a game model produced using a game meta-model can be mapped to a game software model during the process of model transformation in a model driven game development framework.

3.4. Game Software Models

A game software model, as opposed to a game meta-model, represents computer games from a computational perspective independent of any technology platform. From an MDE perspective, it is translated from the game model to specify the data, methods, processes and technologies that comprises the game software. Depending on the game genre, a computer game may require different types of visual renderer, AI, game physics and other software components to realise the game design. These technologies are usually packaged into a game software framework which is used to produce a variety of different computer games. It is rather challenging to have one game software framework to represent all computer games across different genres due to technical constraints that game software frameworks inherit. Although there are differences in technologies used, the basic structure of computer games software remains similar across different genres. It is therefore crucial that a game software model has a high degree of abstraction to allow it to cater for a range of game genres.

A game software model is dependent on the choice of programming paradigm. Early computer games were programmed with languages such as BASIC and C which supports both unstructured and later structured programming paradigms. Today most game developers would opt to engineer games software using an OO model as it offers versatility and reusability of code which can increase productivity and ease maintenance of code. Most OO-influenced models represent game software based on its processes or game components. The *SharpLudus Ontology* (Furtado & Santos, 2006a) is a game software model built onto six key concepts that represent computer games, namely *configuration*, *graphics*, *entities*, *event*, *flow* and *audio*. Each of these concepts is elaborated in detail to represent the properties and methods that construct a 2D adventure game. Although SharpLudus Ontology is designed specifically for the 2D adventure game genre, the software model can also be used for other 2D games that are built using tile-based worlds.

The *GameSystem, DecisionSupport and SceneView* (GDS) model (Sarinho & Apolinário, 2009) represents game software with a combination of features organised collectively. Each feature describes generic configurations and behavioural aspects for a specific part of the game software. Features in GameSystem monitor both game events and inputs from the game-player, and trigger the associated actions that can affect all defined data including those defined in DecisionSupport and SceneView. DecisionSupport groups a selection of game AI features that provides the intelligence for Non-Player Characters (NPC), while SceneView describes the features that observe events within a defined spatial environment and those affecting the presentation of the game such as audio, graphics and physics. This shares many characteristics of the MVC (Model-View-Controller) design pattern.

Altunbay, Cetinkaya, & Metin's (2009) *board game model* is a representation of game software specific to board games. The *board game model* is based on the concepts of: *Game Engine*, *Game Element*, *Player*, *Event*, *Action*, *Game State*, *Goal*, *Sub-Goal*, *Non-Movable Element*, *Movable Element* and *Rules*. Most of these concepts are very specific towards the board game genre except for *Player*, *Goal* and *Rules* which are generic for all games genres.

The SharpLudus Ontology, board game model and GDS each have their own merits and focus in representing games software. The key concepts introduced in SharpLudus Ontology are a useful guide for representing other forms of game software. It represents the essential components in game software but it will require further specific and detail elaboration that reflects the requirements for other game software before it is deemed useful. The board game model is comprehensive but rather specific to a single game genre thereby limiting its usefulness. The GDS model is a credible candidate as it generically and comprehensively represents game software based on features which can be mapped across to a game software framework and therefore can represent a broader range of computer games software. Moreover, it clearly separates games software into MVC components (GameSystem as the model, SceneView as the view and DecisionSupport as the controller) model permitting independent development, testing and maintenance of each component. It is apparent that architecting a game software model requires good understanding of game processes and the game software framework. As current game development trends sets such a high-dependency on game software frameworks to increase productivity, it is important that a game software model structures game design requirements logically and maps to common functionalities of a game software framework.

3.5. Game Software Frameworks

A game software framework (also known in the game industry as a game engine) is a set of Application Programming Interfaces (APIs). It consists of software components that perform 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. An appropriate API can increase the productivity of game developers allowing them to reuse proven and tested software code to produce a variety of computer games within the scope of the software framework. Game software frameworks vary depending on technology features that are often constrained by particular game genres, technology platforms (hardware) and visual dimension of the game world (2D or 3D), but technology components such as graphics, sound, user input, basic game physics and user interfaces remain essential across all game software frameworks. For instance, the *Unreal engine*[4] is suitable for development of action or adventure games in first or 3rd person view but it will require modification to include necessary physics code if it to be used to support sports game genres. Not all game software frameworks support all features since integrating these technological components under

[4] Read more about Unreal Engine from http://www.unrealtechnology.com/.

a single framework is a highly complex and expensive task. Furthermore, not all computer games software will require the entire collection of software components to operate. Budget, support, features offered, ease of use, suitability, royalties and targeted hardware platform are some examples of key criteria which can help to decide the most suitable game software framework for a project. Identifying features supported and techniques used in game software frameworks can help to provide valuable information for MDE developers to architect a framework that is robust and interoperable.

There are many game software frameworks developed to date (at the time of writing, there are 297 3D game engines registered at the DevMaster.net[5] website). In general, these game software frameworks can be categorised from a purely financial point of view as being one of *commercial, open source* and *proprietary*, or using the visual dimension they can cater for i.e. 2D or 3D (see Figure 14).

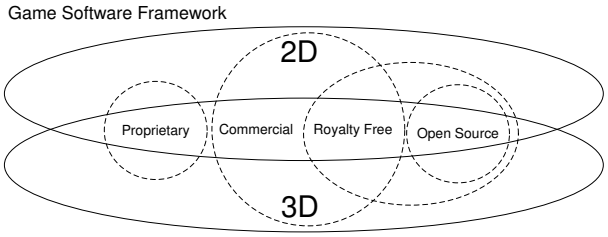

Figure 15: Categories of Game Software Framework

Commercial game software frameworks are much more stable and offer a reliable (supported) option over open source game software frameworks. The *Torque Game Engine* (TGE) is an example of royalty-free game engine which is offered to developers at a very affordable price compared to other commercial game software frameworks, in which TGE developers only pay for the technology once and are allowed to develop as many games software applications as they wish. *Unreal engine*, *Source engine*, *CryEngine* and *Id Tech* (formerly known as *Quake Engine*) are some notable examples of highly sophisticated game technologies that can help developers to produce high-quality games. These premium game technologies are available for licensing and require a one-off royalty payment (per-title fee) or royalty for every copy of game sold, or both. However, some of these are made available for free for educational purposes such as the *3DSTATE 3D Engine*, *XNA*, *CryEngine 3*, *Unreal Development Kit* (UDK) and *Unity 3D*.

Open source game software frameworks are developed for very specific purposes and often made available to the public when they reach the end of their useful lifecycle. Examples of well known open source game software frameworks include *Apocalyx Engine, Baja Engine, Blender Game Engine, Irrlicht Engine, jME, jPCT, Lilith3D, Object Oriented Graphic Rendering Engine* (ORGE), *OpenSceneGraph, Panda3D* and *The Nebula Device 3*. Most of these game software frameworks are distributed under an open source license which allows game developers to produce games software for both commercial and non-commercial purposes at no cost. Some open source game software frameworks only implement a limited range of techniques and some fail to address certain features, for example *Panda 3D* does not provide solution shadows, scene management and curves. Game developers can still achieve the shadows effect using *Panda 3D* either by adding this feature to the framework themselves (this however requires a specialist skill) or by creatively using textures and shaders to achieve the desired shadow effects. However, there is little or sometimes no support at all for customising some technologies to fit the use of a project with a different specification. For more details of these game software frameworks mentioned refer to Table 1.

Table 1: Details on Commercial and Open source Game Software Framework

Game Engine	License	URL
3DSTATE 3D	Open source	http://www.3dstate.com/
Unreal Engine	Commercial & the Unreal Development Kits (UDK) available free for educational purpose	http://www.unrealtechnology.com/ http://www.udk.com/
Source Engine	Commercial	http://source.valvesoftware.com/index.php.
CryEngine	Commercial & available free for educational purpose.	http://www.crytek.com/
Id Tech (formally known as Quake Engine)	Commercial	http://www.idsoftware.com/business/idtech3/
XNA	Commercial & available free for educational purpose	http://www.xna.com/
Unity 3D	Commercial & available free for educational purpose	http://unity3d.com/unity
Apocalyx Engine	Open source	http://apocalyx.sourceforge.net/
Baja Engine	Open source	http://www.bajaengine.com/
Irrlicht Engine	Open source	http://irrlicht.sourceforge.net/
Java Monkey Engine (jME)	Open source	http://www.jmonkeyengine.com/
jPCT	Open source	http://www.jpct.net/
Lilith3D	Open source	http://www.grinninglizard.com/lilith/
Oriented Graphic Rendering Engine (ORGE),	Open source	http://www.ogre3d.org/
OpenSceneGraph	Open source	http://www.openscenegraph.org/projects/osg
Panda3D	Open source	http://www.panda3d.org/
The Nebula Device 3	Open source	http://nebuladevice.cubik.org/

Proprietary game software frameworks are created in-house (by game development companies) solely for internal usage. These are usually on par with commercial game software frameworks and often of more superior quality. *RenderWare*, technology produced by Criterion Games and acquired by Electronic Arts (EA) is used in many games published by EA such as the Need For Speed and Burnout series; *EGO* developed in-house at Codemasters in the UK is used in production of a range of game titles including Colin McRae: Dirt, Race Driver: Grid and Operation Flashpoint: Dragon Rising; *Jade Engine* Ubisoft's in-house game technology is used in games such as Rayman Raving Rabbids I and II, and Teenage Mutant Ninja Turtles. These technologies are mainly used for in-house game titles and are therefore not easily applicable to development of educational or serious games. Based on the ratings gathered from the devmaster.net forum and our analysis on features offered by each game software framework, we rank

[5] See known 3D game engine listing at http://www.devmaster.net/engines/list.php.

these game software frameworks based on popularity and sophistication of features as shown in Figure 15 to give a general idea of the current technology landscape. It is apparent that popular game software frameworks such as Panda3D, XNA, Unity 3D, CryEngine 3 do not necessarily offer more features compared to the less popular game software frameworks like the 3DSTATE 3D Engine and the Nebula Device 3. Some of the reasons these less-sophisticated game software frameworks gain popularity amongst the developers includes the range of technical support, availability of high-level toolsets, large community base for exchange of knowledge and stability of the game software framework.

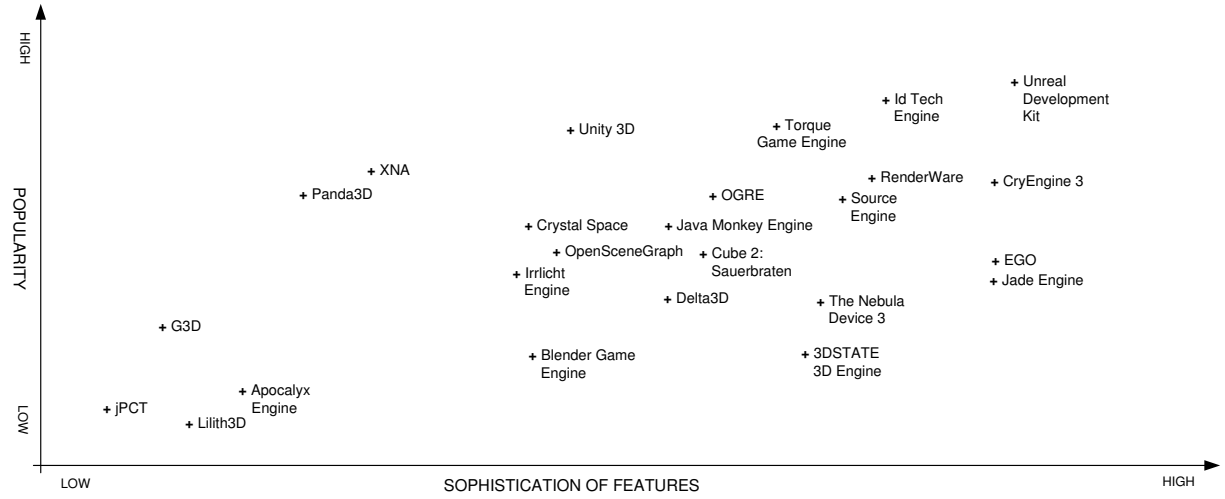

Figure 16: Game Software Frameworks ranked according to Popularity and Sophistication of Features

One of the most crucial components for game software is the graphics which provide the visualisation of the game world to game players. Clever usage of graphic technologies can captivate a game player's interest and provide an immersive game play experience. In general, aspects related to graphic technologies can be categorised into lighting, shadows, texturing, shaders, scene management, meshes, terrain, special effects (particle systems) and rendering. Each of these aforementioned categories has its own collection of specialised techniques to achieve the desired visual results without compromising the frame rate (see Table 2). In fact most of the game software frameworks are essentially real-time 3D graphics engines that have been extended to form a game software framework. For example, game physics engines such as the Newton Game Dynamics[6], Open Dynamic Engine[7] or Havok Physics[8], game AI engines such as Havok AI[9] and network components for games such as Game Networking Engine[10] (GNE) or RakNet[11] can be added complimentarily to any 3D graphics engine. The remaining features which are not available can be developed in-house by specialist game software engineers to complete the game software framework. Physics, AI, animation, networking and sound have a variety of techniques available to support game development (see Table 3). In addition, some game software frameworks provide a collection of tools such as a level-editor, visual sound tools and animation tools to allow non-technical members of the team to fine-tune games visually instead of directly editing the source code. This review of game software framework features and their associated techniques provides an overview of the level of support a game software model should offer to support a range of game platforms. Therefore it is crucial that any game software model can be extended easily over time to remain heterogeneous among the game software frameworks. To achieve this design requirement, it is necessary that the game software model be highly abstracted else it can only support a limited number of game software frameworks thereby constraining the variety of educational or serious games which can be built with future game software frameworks.

Table 2: List of Techniques for Graphic Features.

Graphic Features	Techniques
Lighting	Light mapping, gloss mapping, per-pixel lighting, per-vertex lighting, anisotropic filtering and radiosity.
Shadows	Shadowmap, projected planar and shadow volume.
Texturing	Basic, multi-texturing, bump mapping, mip mapping, volumetric texturing, projective texture mapping and procedural texturing.
Shaders	Vertex shader, pixel shader and high-level shader.
Scene Management	General, Binary Space Partition (BSP), Potentially Visible Set (PVS), portals, occlusion culling, octrees and Level of Details (LOD).
Meshes	Mesh loading, mesh skinning, progressive mesh, tessellation, deformation
Terrain	Rendering, Continuous Level of Details (CLOD) and splatting.
Special Effects	Environment mapping, billboard, sky, water, fog, mirror, lens flares, particle systems, fire, explosion, decals, weather, motion blur and depth of field.
Surfaces & Curves	Spline and Patches.
Rendering	Fixed-function, stereo rendering, render-to-texture, fonts and Graphical User Interface (GUI).

[6] Read more about Newton Game Dynamics from http://newtondynamics.com/forum/newton.php.
[7] Read more about Open Dynamic Engine from http://www.ode.org/.
[8] Read more about Havok Physics from http://www.havok.com/index.php?page=havok-physics.
[9] Read more about Havok AI from http://www.havok.com/index.php?page=havok-ai.
[10] Read more about GNE from http://www.gillius.org/gne/.
[11] Read more about RakNet from http://www.jenkinssoftware.com/raknet/index.html.

Table 3: List of Techniques for Other Game Feature.

Other Features	Techniques
Physics	Basic physics, collision detection, rigid body and vehicle physics.
AI	Scripted, path-finding, finite state machines, decision making, neural network, fuzzy logic and etc.
Animation	Keyframe Animation, inverse kinematics, forward kinematics, skeletal animation, facial animation, animation blending and morphing.
Network	Peer-to-peer, master-server, client-server, communication (text, voice and video)
Sound	2D sound, 3D sound, streaming sound, mixing, digital signal processing effects and filtering.

3.6. Model-Driven Game Development Frameworks

Supporting any MDE practice is a framework that unifies models and manages the transformation between these models using appropriate MDE tools to produce the desired software artefact. The OMG's Model Driven Architecture (MDA)(OMG, 2001), the Domain-Driven Software Development Framework (Agrawal, Karsai, & Ledeczi, 2003) and Modelling Turnpike (mTurnpike) (Wada & Suzuki, 2006) are examples of model-driven frameworks that aid software architects to develop their own MDE solution to suit a particular domain. These model-driven frameworks have been adapted to suit domains such as security (Basin, Doser, & Lodderstedt, 2006), content repurposing (Obrenovic, et al., 2004), software testing (Javed, Strooper, & Watson, 2007) and pervasive computing (Hemme-Unger, Flor, & Vogler, 2003).

In the game development domain, the use of software frameworks and tools are usual practice amongst commercial game developers. Although current practice does improve productivity of the development team while providing maximum control and flexibility to artistically craft the game software, the production pipeline is still very reliant on specialist artists and programmers. This is rarely an issue in the commercial sector where budgets are supported by a business case. However in the case of game-based learning, the initial cost may be an issue but nonetheless MDE can help to minimise the reliance on professional game developers and assist non-technical domain experts to produce customised educational games through the use of high-level tools.

At present, research in this area is still in its infancy. The *SharpLudus Game Software Factory* (Furtado & Santos, 2006b) is an early attempt of a model-driven approach to increase productivity of game development teams in developing 2D adventure games. The framework consists of (Furtado, 2006);

- A domain-specific modelling language - *SharpLudus Game Modelling Language* (SLGML) that allows the game designer to model the flow of the game using a *room designer* and *info display designer*;
- A semantic validator that checks the model to ensure the design conforms to the semantics of SLGML, and finally;
- A code generator built on top of Microsoft's Visual Studio Integrated Development Environment (IDE) targeted towards generation of C# code for the XNA game engine.

Reyno & Cubel's (2008) *Model-Driven Game Development* (MDGD) approach introduces the use of a selection of UML diagrams to gather required information to automate generation of code for 2D platform games. The framework comprises:

- Two Platform Independent Models (PIM); A UML Class Diagram extended with stereotypes is used to model the relationship between different game entities[12] within the game world while a UML State Transition Diagram is use to model behaviour of the game entity;
- A Platform Specific Model (PSM) to map game actions to hardware controls, and;
- A transformation tool to translate the models into C++ source code compliant to the Haaf Game Engine[13].

Altunbay, Cetinkaya, & Metin's (2009) model-driven approach for developing board games shares some similarity with the MDGD approach. It uses the UML class diagram as the modelling language to represent the game model. The framework is comprised of:

- A *Board Game Meta-model* represented using a UML Class Diagram;
- A *Game Domain Specific Language*[14] (GameDSL) - a meta-model developed specifically to aid the definition of game logic;
- A model-to-model transformation tool to transform the Board Game Model onto GameDSL and model-to-text transformation tool which subsequently transforms the Board Game Model in GameDSL format to Java source code.

The SharpLudus Game Software Factory, MDGD and Board Game Framework are research-driven attempts to adopt a model-driven approach towards developing 2D games. These frameworks may seem to share similar architectures but they differ in terms of models, tools and targeted platform. These frameworks are also developed for a specific game genre and therefore adapting these frameworks to suit other genres would require redefinition of models and new generators to facilitate transformation of model to code. It is apparent that a model-driven educational game framework that can support multiple game software frameworks and is easily extensible is highly desirable to drive the production of a range of educational games on different media. Architecting such a framework to bind both the game model and game software model is a challenge and will require extensive research on game design, game development and a strong understanding of game software frameworks.

3.7. Model-Driven Game Development Environments

Model-driven game development environments are a realisation of model-driven game development frameworks and associated toolset that allow game developers to take advantage of these technologies to accelerate game development. These tools may resemble some of the tools used by the industry (for example, a game editor or level editor) both in functionality and interface but are presented as simplified interfaces for the game designer to describe the game models in a structured and orderly manner.

As described in Section 3.6, the SharpLudus Game Software Factory offers a range of tools to assist game designers in modelling different aspects of 2D adventure games. The development environment is a customised version of the Microsoft Visual Studio IDE that allows game designers to systematically define the game through a collection of wizard-based user interfaces. The *SLMGL Visual Editor* provides a visual drag-and-drop environment for modelling game flow and a range of form-based user interfaces to create the representation of game entities, design the screen, design the layout of the room (a game level), group collections of images as sprites, link audio files, and define conditions for game states, events and event feedback in the game. These interfaces are viewports into the game model which allows game designers to effectively concentrate on a specific aspect of the game to avoid information overload. To ensure game models conform to the semantics of SLGML, a semantic validator is incorporated into the development environment to alert designers about errors in modelling before code is generated. The SharpLudus Game Software Factory uses the *Microsoft Domain Specific Language (DSL) Tools*[15] to implement its code generator by scripting the transformation rules to map the model to generate the code compliant to the game software framework used in the project (Furtado, 2006).

[12] Game entity refers to objects within the game world. This term is synonymous with game object in this paper.
[13] Read more about Haaf Game Engine from http://hge.relishgames.com/.
[14] Read more about GameDSL from http://gamedsl.tuxfamily.org/.
[15] Microsoft DSL Tools is a visualisation and modelling software development kit made available only for the Microsoft Visual Studio IDE to enable software engineers to build custom visual modelling environment to be hosted within the Microsoft Visual Studio IDE. Read more about Microsoft DSL Tools from http://code.msdn.microsoft.com/DSLToolsLab.

The game development environment used by Reyno & Cubel (2008) is implemented using the *Eclipse Modelling Framework*[16] (EMF) which allows the user to model 2D platform games using UML diagrams. The modelling tool provides the standard UML notations to model structure in the form of a class diagram, behaviour of a game entity using a state diagram and control mapping using an activity-like diagram. Transformation from these diagrams to code is implemented using *MOFScript*[17] to generate C++ code compatible with the *Haaf Game Engine*. *MOFScript* is a tool for text transformation that generates text or code from any MOF-based model such as UML.

Altunbay, Cetinkaya, & Metin (2009) use a similar approach as Reyno & Cubel in using a modelling environment built on EMF. A board game model is then translated to GameDSL using the *Atlas Transformation Language*[18] (ATL) model transformation tool and finally transformed into Java source code using *MOFScript*.

Amongst the reviewed model-driven game development environments, the SLGML modelling environment is a promising example of a game modelling environment. This is mainly due to the choice of DSML and interfaces used. The use of domain specific notations instead of the industry standard UML notations is a better choice as it makes games modelling more relevant to the users. Although UML provides more flexibility in modelling games software, it is rather technical and unguided. Only experienced users who have a good understanding of game software would be able to model a game correctly. It is also important to balance the use of visual notations and other forms of user interface for users to model a game. An appropriate interface can increase productivity and reduce the learning curve of using the development environment. Therefore it is crucial that model-driven educational games development environments use DSML notations suited for the domain and a collection of interfaces that encapsulate the technical aspects of game development from domain experts.

3.8. Model-Driven Engineering Technologies

Integrated environments can provide a complete solution where the practitioner can define a DSL and its transformation rules to generate the desired software artefacts and host the modelling component within the same environment, while a code framework provides the basis for development of MDE tools to support the MDD.

3.8.1. Integrated Environments

MetaCase MetaEdit+[19] is a commercial integrated environment for MDE. It provides a collection of tools to design a DSML by defining the concepts, constraints associated to the concepts, symbols representing the concepts and a generator to produce the desired software artefacts with *MetaEdit+ Workbench*. Modelling using the defined DSML is hosted within the MetaEdit+ environment. The *MetaEdit+ Model* extends the generic modelling environment to provide a range of different views allowing the development team to view and edit or manipulate the model as a diagram, matrix or table. In addition, it provides the necessary tools for management of the model and its contents and provides support for a multi-user environment.

Microsoft DSL Tools is another integrated environment for MDE made available as an extension to the *Microsoft Visual Studio IDE* (Cook, et al., 2007). It provides the necessary tools for defining a DSML, modeller and generator to generate textual artefacts from the model within the Visual Studio environment. In DSL Tools, a domain model is created using a UML-like modelling language and a DSML is then represented with graphical notations consisting of shapes and connectors. Relevant generators can be defined in *Extensible StyleSheet Language Transformation* (XSLT) and text artefacts are generated through the Microsoft XSLT engine or by through a parameterised text template generation approach.

Visual Paradigm's Smart Development Environment[20] (SDE) is another commercially developed plug-in to support MDE practices which offers capabilities to model in UML, entity relationship diagram, data flow diagram and others, generate code for 15 different programming languages, team collaboration capabilities, round-trip engineering and can be integrated with well-known IDEs such as Visual Studio, NetBeans, Eclispe and intelliJ IDEA. The modelling environment is hosted within the supported IDE and code is instantly generated in the code editor view allowing both forward and reverse engineering.

Apart from commercially driven integrated environments for MDE, there are also non-commercial products available. GME, developed at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University, provides a similar facility for defining a DSML and hosting modelling within the same environment (Ledeczi, et al., 2001). In GME, a domain model is modelled in UML and assigned with graphical representation while modelling rules are defined using *Object Constraint Language* (OCL). Models are then saved directly in XML format or translated using an external interpreter (generator) into other textual artefacts using *Microsoft Common Object Model* (COM) technology through extensions to the GME.

GEMS[21] is a meta-modelling tool developed by ISIS at Vanderbilt University that provides GME-like meta-modelling to create modelling tools that can be hosted within the Eclipse environment. It uses Eclipse modelling technologies such as EMF and *Graphical Modelling Framework* (GMF). Like *GME*, a DSML is created in *GEMS* using UML-like notations and later ported automatically to Eclipse as plug-in. Generators and transformation engines can be created for the modelling environment created using *Open Architecture Ware*[22] (OAW), *Java Emitter Templates*[23] (JET), and *Atlas Transformation Language*[24] (ATL) to transform the model into desired textual artefacts.

The *NetBeans IDE*[25] also provides similar facilities for modelling using a *UML plug-in*[26] and code generation using a template engine such as *FreeMarker*[27]. The *UML plug-in* enables the modelling environment to model the SUS using a range of supported UML diagrams such as activity diagram, class diagram, sequence diagram, state diagram, and use case diagram as a Java platform model or PIM. The *FreeMarker* code templates associated with the diagrams can then be customised to generate the desired Java source code.

Among the reviewed integrated environments for MDE, MetaCase MetaEdit+ offers comprehensive and customisable modelling features, which can be integrated into many IDEs, and it offers a facility to create generators for any text artefacts. GME is also a standalone tool with a similar aim, but suffers from the lack of a built in generator. For Visual Studio users, Microsoft DSL Tools is an alternative to MetaEdit+ where a DSML can be created, hosted and generate code entirely within the Visual Studio environment. Similarly, GEMS provides a facility to create the modelling environment as an add-on to the Eclipse IDE with support to create generators for any text artefacts. Each of the reviewed integrated environments provides the facility to support the MDE process in one way or the other. The SDE allows users to model using UML diagrams, entity relationships diagram, BPMN, process map and others. It also provides great support for code generation and integration with well known IDEs. The NetBeans IDE with the UML plug-in is an open source alternative to SDE. Selection of the integrated environment will ultimately depend on the available budget to invest and choice of development tools. These MDE integrated environments can be used for development of a model-driven educational game development environment similar to those proposed in Section 3.7 and realisation of such a model-driven educational game development environment will require a model-driven framework that supports development of educational games similar to those described in Section 3.6.

[16] Read more about EMF from http://www.eclipse.org/modeling/emf/.
[17] Read more about MOFScript from http://www.eclipse.org/gmt/mofscript/.
[18] Read more about ATL from http://www.eclipse.org/atl/.
[19] Read more about MetaEdit+ from http://www.metacase.com/.
[20] Read more about SBE from http://www.visual-paradigm.com/product/sde/.
[21] Read more about GEMS from http://www.eclipse.org/gmt/gems/.
[22] Read more about OAW from http://www.eclipse.org/workinggroups/oaw/.
[23] Read more about JET from http://www.eclipse.org/modeling/m2t/?project=jet.
[24] Read more about ATL from http://www.eclipse.org/m2m/atl/.
[25] Read more about NetBeans IDE from http://netbeans.org/.
[26] Read more about UML Plugin for NetBeans IDE from http://netbeans.org/features/uml/.
[27] Read more about FreeMarker from http://freemarker.sourceforge.net/.

3.8.2. Code Frameworks

Code frameworks for MDE can generally be divided into two categories; *diagramming* and *transformation*. IBM's *ILOG Visualisation*[28] and the *Eclipse Graph Editing Framework*[29] (GEF) are examples of diagramming code frameworks that provide the software components such as 2D graphics, notation palette, environment layout support, viewer, connection anchoring, connection routing, connection decoration and cursor support which are necessary for creation of a diagramming environment. ILOG Visualisation is licensable as GUI components for the .NET, Java, C++ and Adobe Flex platforms, whereas GEF is only available as a plug-in on the Java platform specifically for the Eclipse IDE.

The *Eclipse Modelling Framework* (EMF), *CodeWorker*[30] and ATL are examples of transformation code frameworks. The EMF provides the facility to build a code generator in the Eclipse IDE. It accepts a model expressed in annotated Java, XML or *XML Metadata Interchange* (XMI) and transforms it into Java implementation classes or a formatted XML document. An alternative to EMF is CodeWorker which is freely available as a Java interface, .NET assembly and an Eclipse plug-in. For model-to-model transformation, the ATL can be used to transform a set of source models to a set of target models. The ATL is offered as plug-in to the Eclipse IDE.

The Eclipse *Graphical Modelling Framework*[31] (GMF) provides a full-featured modelling environment by integrating the diagramming features from GEF and the transformation features from EMF. This reduces the complication for developers who want to create a full-featured modelling environment in the Eclipse IDE. For non-Eclipse users, modelling environments can be built on the integration of ILOG Visualisation and CodeWorker. Building MDE tools based on these code frameworks described can be time consuming compared to the use of an integrated environment described in Section 3.8.1. However, it provides great flexibility for developers to create custom interfaces and target code generation to required platforms more accurately.

3.9. Assistive User Interfaces

The game modelling environments reviewed in Section 3.6 use a range of user interfaces to capture game requirements from users. Diagramming interfaces that use shapes (or images) and connectors, wizards, visual editors and text editors are standard approaches found in game modelling environments. Each of these approaches provides a different form of support to users in defining the game world as some require a visualisation environment and some do not. Selection of the right interface is crucial especially in the view of this research study as the primary users are domain experts who may not have much technical knowledge in games development.

Assistive user interfaces provide guidance to users in preparing a model. Software based 'wizard' interfaces are a simple way to guide users through systematic information entry. However, they require a number of custom 'wizard' interfaces to enable users to enter details of a model during modelling and do not provide a high-level view of the model. Diagramming interfaces and visual editors provide some form of guidance through visual cues and feedback from interactions. They do, however, require users to undergo some form of training or tutorial before they become proficient in using the tool. Among all user interfaces, the text editor provides the least guidance in entering formal information as it requires statements to be entered that conform to the syntax of some formal language. Although modern text editors include syntax colouring, code tooltips, error highlighting and many other advanced facilities built-in, users are still prone to committing errors syntactically.

Alice (Kelleher, et al., 2002) and Scratch (Maloney, et al., 2004) are examples of applications that use assistive user interfaces developed to address the needs of non-technical users. Alice is an educational software development tool that motivates and aids students in learning computer programming within a 3D environment. Understanding the frustration of students learning programming, Alice 2.2 and Alice Storytelling are based on a drag and drop interface that addresses the major problem of many learners i.e. syntax errors (Kelleher, et al., 2002). Commands, programming constructs, 3D objects, object properties, and behaviour are represented as tiles with different colours which users can drag into the animation and behaviour areas with a menu that displays the available options for selection of parameters to complete a statement that represents the animation or behaviour of an object in a 3D world (see Figure 16).

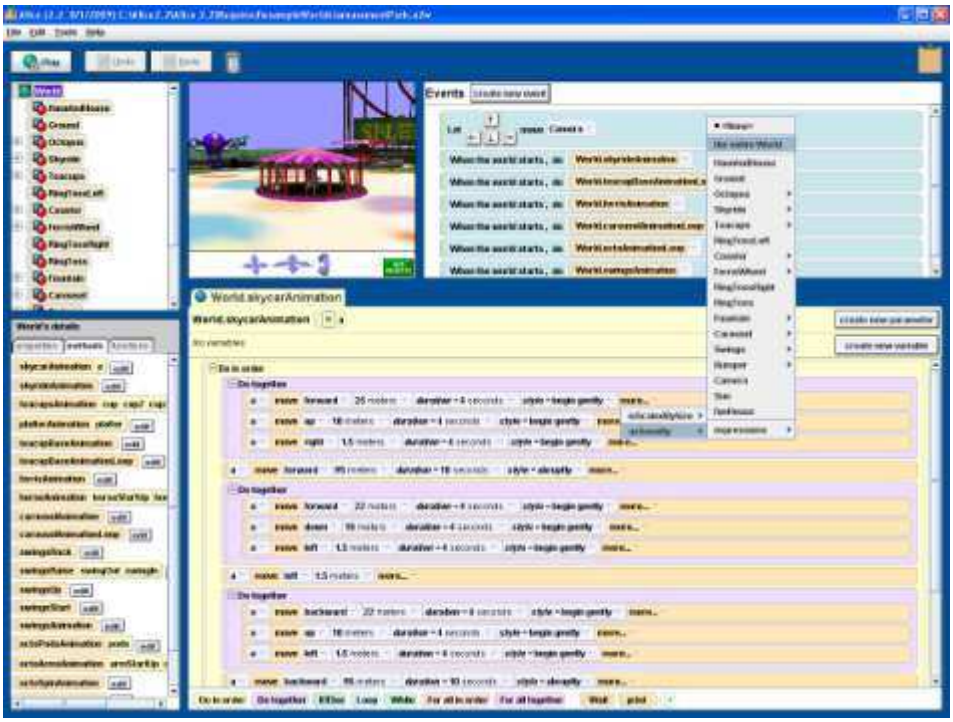

Figure 17: Screenshot of Alice 2.2 User Interface

[28] Read more about IBM ILOG Visualisation from http://www-01.ibm.com/software/websphere/products/visualization/.
[29] Read more about GEF from http://www.eclipse.org/gef/.
[30] Read more about CodeWorker from http://codeworker.free.fr/.
[31] Read more about GMF from http://www.eclipse.org/modeling/gmf/.

Scratch is another initiative similar to Alice developed by the MIT Media Lab to assist children learning programming concepts (Maloney, et al., 2004). It provides a rich media authoring environment with much simpler interfaces and uses a graphical programming language to enable learners to script multimedia objects and interactivity. Programming constructs are grouped into motions, controls, looks, sensing, sound, operators, pen and variables using puzzle-style coloured block as notations. Within each construct, values can be inserted via the textboxes or list boxes available as shown in Figure 17. Blocks are fitted together based on the compatibility of the command and data shapes to eliminate syntax error, thus lowering the barrier of learning programming for beginners.

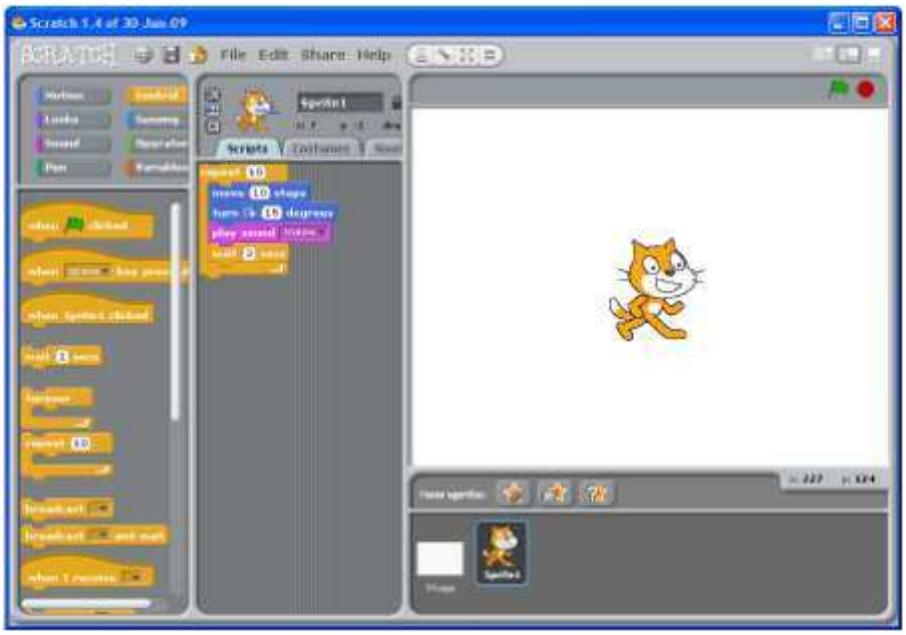

Figure 18: Screenshot of Scratch User Interface

Both the reviewed user interfaces are examples of block-styled drag-and-drop interfaces with clever use of input type to simplify interactions. The constraints introduced in these user interfaces eliminate unnecessary errors and also guide users in completing formal text-entry. In general, such user interfaces can lower the learning curve when using technical tools and should be adapted into model-driven educational game development environments to aid non-technical domain experts in production of educational games.

4. Conclusions

Changes in learner's behaviour are inevitable and many believe that game-based learning is one of the solutions to encourage learners to engage more in the learning process. In addition to the motivating factor of reintroducing play into learning, game-based learning offers an opportunity to embed a wide range of learning approaches. With the current generation of 'digital natives' being exposed to interactive entertainment during childhood years and being brought up in the digital era, we believe that game-based learning will become a preferred method of learning for many of this generation in the future (Tang & Hanneghan, 2010). These beliefs are supported by JISC's "Guide to effective learning design" which states that learning should be congruent to lifestyle for effective learning to take place (JISC, 2004). Indeed there are a growing number of domain experts intending to take advantage of game-based learning and a growing demand for high-level tools which can help them create their own educational games. In this article, we have described how MDE can be used a platform for one such technological solution to the dilemma and outlined the benefits and challenges associated in creating a model-driven educational development environment. Our survey provides insights on developments in this field which plays an important role collectively on the creation of a model driven game development environment. We identified these approaches and solutions, and reviewed them from MDE perspectives. Although most of the current approaches and solutions are related to model driven game development, using these approaches and solutions will require clever modification before they can be deemed suitable to be integrated in a model driven game development framework. Nevertheless, it provides a reference for framework developers to begin with, extend or adapt. Although current efforts in model-driven game development mainly focus on mainstream audiences (i.e. industry and hobbyist), there is a growing awareness of the adoption of MDE specifically for production of educational games and serious games. Developing a model-driven educational games development environment for use by domain experts requires a different treatment compared to present approaches. Pedagogic practices for educational games design (as previously suggested in our earlier work (Tang & Hanneghan, 2010)) should be embedded within the modelling environment to ensure any computer games produced are pedagogically sound. We are currently developing our own game definition model as well as a game software model for use in our own model-driven educational games development framework. Furthering our research efforts we intend to prototype a model-driven educational game development environment using our framework as a proof of concept. At a later stage of the research we aim to provide case studies on the use of the new tool and findings from the evaluation will be used to further improve this tool.

References

Adams, E. (2004). Designer's Notebook: Designing with Gameplay Modes and Flowboards Retrieved 31 October, 2009, from http://www.gamasutra.com/view/feature/2101/designers_notebook_designing_.php

Adams, E., & Rollings, A. (2006). *Fundamental of Game Design*. US: Prentice Hall.

Agrawal, A., Karsai, G., & Ledeczi, A. (2003). *An end-to-end domain-driven software development framework*. Paper presented at the Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.

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

Altunbay, D., Cetinkaya, E., & Metin, M. G. (2009, 20th May). *Model Driven Development of Board Games* Paper presented at the First Turkish Symposium on Model-Driven Software Development (TMODELS), Bilkent University, Ankara Turkey.
Ang, C. S., & Rao, G. S. V. R. K. (2004). Designing Interactivity in Computer Games a UML Approach. *International Journal of Intelligent Games and Simulation, 3*(2), 62-69.
Basin, D., Doser, J., & Lodderstedt, T. (2006). Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol., 15*(1), 39-91.
Bates, B. (2004). *Game Design* (2 ed.). Boston, MA: Thomson Course Technology PTR.
BECTa (2006). Computer Games in Education: Findings Report Retrieved 19 February, 2008, from http://partners.becta.org.uk/index.php?section=rh&rid=13595
Bethke, E. (2003). *Game Development and Production*. Plano, Texas: Wordware Publishing.
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.
Björk, S., & Holopainen, J. (2004). *Patterns in Game Design*: Charles River Media.
Björk, S., Lundgren, S., & Holopainen, J. (2003, 4 - 6 November). *Game Design Patterns*. Paper presented at the Level Up, University of Utrecht, The Nethelands.
Ceri, S., Daniel, F., Facca, F. M., & Matera, M. (2007). Model-driven Engineering of Active Context-awareness. *World Wide Web, 10*(4), 387-413.
Checkland, P., & Scholes, J. (1990). *Soft Systems Methodology in Action*. West Sussex, England: John Wiley & Son.
Cook, S., Jones, G., Kent, S., & Wills, A. C. (2007). *Domain-Specific Development with Visual Studio DSL Tools*. Crawfordsville, Indiana: Addison Wesley.
Crawford, C. (1982). *The Art of Computer Game Game Design*.
El-Rhalibi, A., Hanneghan, M., Tang, S., & England, D. (2005). *Extending Soft Models to Game Design: Flow, Challenges and Conflicts*. Paper presented at the DiGRA 2005 - the Digital Games Research Association's 2nd International Conference.
FAS (2006). Harnessing the power of video games for learning, Summit on Educational Games 2006, from http://fas.org/gamesummit/Resources/Summit%20on%20Educational%20Games.pdf
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.
Fondement, F., & Silaghi, R. (2004). *Defining Model Driven Engineering Processes*. Paper presented at the 3rd Workshop in Software Model Engineering (WiSME2004).
France, R., & Rumpe, B. (2007). *Model-driven Development of Complex Software: A Research Roadmap*. Paper presented at the 29th International Conference of Software Engineering: Future of Software Engineering.
Fullerton, T. (2008). *Game Design Workshop, Second Edition: A Playcentric Approach to Creating Innovative Games* (2 ed.). Burlington, MA: Morgan Kaufmann.
Furtado, A. W. B. (2006). *SharpLudus: Improving Game Development Experience Through Software Factories And Domain-Specific Languages.* Federal University of Pernambuco, Pernambuco, Brazil.
Furtado, A. W. B., & Santos, A. L. M. (2006a). *Defining and Using Ontologies as Input for Game Software Factories*. Paper presented at the 5th Brazilian Symposium on Computer Games and Digital Entertainment.
Furtado, A. W. B., & Santos, A. L. M. (2006b). *Using Domain-Specific Modeling towards Computer Games Development Industrialization*. Paper presented at the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06).
Futurelab, N. (2006). Close to 60% of UK Teachers Want Computer Games in the Classroom Retrieved 30 June, 2008, from http://www.futurelab.org.uk/about_us/Press_Release184
Gal, V., Prado, C. L., Natkin, S., & Vega., L. (2002). *Writing for video games*. Paper presented at the VRIC 2002.
Grønmo, R., Skogan, D., Solheim, I., & Oldevik, J. (2004). *Model-driven Web Services Development*. Paper presented at the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04).
Harel, D., & Kugler, H. (2001). The Rhapsody Semantics of Statecharts (or, On the Executable Core of the UML). *Lecture Notes in Computer Science, Integration of Software Specification Techniques for Application in Engineering*(3147), 325 - 354.
Hemme-Unger, K., Flor, T., & Vogler, G. (2003). *Model driven architecture development approach for pervasive computing*. Paper presented at the Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.
Heskett, J. (2005). *Design: A Very Short Introduction*. New York: Oxford University Press.
Hildenbrand, T., & Korthaus, A. (2004). *A Model-Driven Approach to Business Software Engineering*. Paper presented at the 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2004).
Hill, J. H., Tambe, S., & Gokhale, A. (2007). *Model-driven Engineering for Development-time QoS Validation of Component-based Software Systems*. Paper presented at the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07).
Hunicke, R., LeBlanc, M., & Zubek, R. (2004). *MDA: A Formal Approach to Game Design and Game Research.* Paper presented at the Challenges in Game AI Workshop, AAAI04.
Järvinen, A. (2007, 24 - 28 September). *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.
Javed, A. Z., Strooper, P. A., & Watson, G. N. (2007). *Automated Generation of Test Cases Using Model-Driven Architecture*. Paper presented at the Proceedings of the Second International Workshop on Automation of Software Test.
Jenkins, H., Klopfer, E., Squire, K., & Tan, P. (2003, October 2003). Entering The Education Arcade. *Computers in Entertainment (CIE), 1,* 17-17.
JISC (2004). *Effective Practice with e-Learning - A good practice guide in designing for learning*.
Kelleher, C., Cosgrove, D., Culyba, D., Forlines, C., Pratt, J., & Pausch, R. (2002). *Alice2: Programming without Syntax Errors* Paper presented at the User Interface Software and Technology.
Kelly, H., Howell, K., Glinert, E., Holding, L., Swain, C., Burrowbridge, A., et al. (2007). How to build serious games. *Communications of the ACM, 50*(7), 44-49.
Kelly, S., & Tolvanen, J.-P. (2008). *Domain-Specific Modeling: Enabling Full Code Generation*. Hoboken, New Jersey: Wiley-IEEE Computer Society Press.
Kienzle, J., Denault, A., & Vangheluwe, H. (2007). Model-Based Design of Computer-Controlled Game Character Behavior In G. Engels, B. Opdyke, D. C. Schmidt & F. Weil (Eds.), *Lecture Notes in Computer Science* (Vol. 4735/2007, pp. 650-665): Springer Berlin / Heidelberg.
Kleppe, A. G., Warmer, J. B., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture : Practice and Promise*: Addison-Wesley.
Ledeczi, A., Maroti, M., Bakay, A., Karsa, G., Garrett, J., Thomason, C., et al. (2001). *The Generic Modeling Environment.* Paper presented at the Workshop on Intelligent Signal Processing, Budapest, Hungary.
Lodderstedt, T., Basin, D., & Doser, J. (2002). *SecureUML: A UML-Based Modeling Language for Model-Driven Security*. Paper presented at the Model Engineering, Concepts, and Tools 5th International Conference.
Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., & Resnick, M. (2004). *Scratch: A Sneak Preview.* Paper presented at the Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing.
Natkin, S., Vega, L., & Grünvogel, S. M. (2004). *A New Methodology for Spatiotemporal Game Design*. Paper presented at the International Conference on Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004).

Nelson, M. J., & Mateas, M. (2007). *Towards Automated Game Design.* Paper presented at the 10th Congress of the Italian Association for Artificial Intelligence on AI*IA: Artificial Intelligence and Human-Oriented Computing, Rome, Italy.
Obrenovic, Z., Starcevic, D., & Selic, B. (2004). A Model-Driven Approach to Content Repurposing. *IEEE Multimedia, 11*(1), 62-71.
OMG (2001, 21 December 2008). Model Driven Architecture (MDA), from http://www.omg.org/docs/ormsc/01-07-01.pdf
Onder, B. (2002). Writing Adventure Game. In F. o. D. Laramée (Ed.), *Game Design Perspectives* (pp. 28-43). Hingham, Massachusetts: Charles River Media.
Pearce, C. (2006). Productive Play: Game Culture From the Bottom Up. *Games and Culture, 1*(1), 17-24.
Reyno, E. M., & Cubel, J. Á. C. (2008). *Model-Driven Game Development: 2D Platform Game Prototyping.* Paper presented at the GAMEON' 2008.
Rollings, A., & Adams, E. (2003). *Andrew Rollings and Ernest Adams on Game Design*: New Riders Publishing.
Rollings, A., & Morris, D. (2004). *Game Architecture and Design: A New Edition.* Indianapolis, Indiana: New Riders.
Rouse, R. (2001). *Game Design: Theory & Practice* (2 ed.). Plano, Texas: Wordware Publishing.
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.
Sarinho, V., & Apolinário, A. (2009). *A Generative Programming Approach for Game Development.* Paper presented at the VII Brazilian Symposium on COmputer Games and Digital Entertainment, Rio de Janeiro, Brazil.
Sawyer, B., & Smith, P. (2008). Serious Games Taxonomy Retrieved 26 March, 2008, from http://www.dmill.com/presentations/serious-games-taxonomy-2008.pdf
Schauerhuber, A., Wimmer, M., & Kapsammer, E. (2006). *Bridging existing Web modeling languages to model-driven engineering: a metamodel for WebML.* Paper presented at the Second international workshop on model driven web engineering (MDWE'06).
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.
Sottet, J.-S., Calvary, G., Favre, J.-M., Coutaz, J., Demeure, A., & Balme, L. (2006). *Towards Model Driven Engineering of Plastic User Interfaces* Paper presented at the MoDELS 2005 International Workshops OCLWS, MoDeVA, MARTES, AOM, MTiP, WiSME, MODAUI, NfC, MDD, WUsCAM.
Tang, S., & Hanneghan, M. (2005). *Educational Games Design: Model and Guidelines.* Paper presented at the 3rd International Game Design and Technology Workshop (GDTW'05).
Tang, S., & Hanneghan, M. (2010). Designing Educational Games: A Pedagogical Approach. In P. Zemliansky & D. Wilcox (Eds.), *Design and Implementation of Educational Games: Theoretical and Practical Perspectives* (pp. 108-125). Hershey, PA: IGI Global.
Tang, S., Hanneghan, M., & El-Rhalibi, A. (2004, November 25-27). *Designing Challenges and Conflicts: A Tool for Structured Idea Formulation in Computer Games.* Paper presented at the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004), Het Pand, Ghent, Belgium.
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.
Taylor, M. J., Baskett, M., Hughes, G. D., & Wade, S. J. (2007). Using Soft Systems Methodology for Computer Game Design. *Systems Research and Behavioral Science, 24*, 359-368.
Taylor, M. J., Gresty, D., & Baskett, M. (2006). Computer game-flow design. *Computers in Entertainment (CIE), 4*(1), Article No. 5.
Thramboulidis, K. C. (2004). Paper presented at the 2nd IEEE International Conference on Industrial Informatics INDIN'04.
Wada, H., & Suzuki, J. (2006). Modeling Turnpike: A Model-Driven Framework for Domain-Specific Software Development *Satellite Events at the MoDELS 2005 Conference* (pp. 357-358).
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, Simon Fraser University, Burnaby, BC, Canada.