A Model-Driven Framework to Support Development of Serious Games for Game-based Learning

Stephen Tang and Martin Hanneghan
School of Computing and Mathematical Sciences,
Liverpool John Moores University
Byrom Street, L3 3AF Liverpool, UNITED KINGDOM.
Email: {o.t.tang, m.b.hanneghan} @ljmu.ac.uk

Abstract—Computer games, predominantly a form of interactive entertainment, have been repurposed for game-based learning in recent years. Many studies have presented the educational value that computer games possess with evidence that supports the positive experiences of game-based learning. Game-based learning could be an invaluable learning approach for the 21st century, but is currently hindered by the lack of availability of serious games to support this innovative approach. Developments in software engineering that enable automatic generation of software artefacts through modelling promises new hope for game-based learning adopters, especially those with little or no technical knowledge, to produce their own serious games for use in game-based learning. This paper presents a new framework based on a model-driven approach designed to aid non-technical domain experts in the production of serious games.

Keywords—Model Driven Engineering, Model Driven Development, Serious Games, Game-Based Learning.

I. INTRODUCTION

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. It refers to the innovative learning approach derived from the use of computer games that possess educational value or different kinds of software applications that use games for learning and education purposes such as learning support, teaching enhancement, assessment and evaluation of learners [1]. Computer games specifically designed for such purposes are generally termed as educational games or serious games which is a term used to describe computer games with embedded pedagogy [2]. It also encompasses games for health, advertisement, training, education, science, research, production and work, in which games technologies are used specifically for improving accessibility of simulations, modelling environments, visualisation, interfaces, delivery of messages, learning and training, and productive activities such as authoring, development or production [3]. In this paper, we use the term serious games to refer to computer games developed for training and educational purposes.

Game-based learning has been advocated by many commentators that it can provide an enhanced learning experience compared to traditional didactic methods. In recent years, great efforts have been put into realising game-based learning as demand for technology assisted learning approaches becomes popular among domain experts and academics and offer greater relevance for the so-called ‘PlayStation’ generation learners.

However, the adoption of such a seductive learning method engenders a range of technical, educational and pedagogical challenges, including: (i) how to enable non-technical domain experts (teachers) - with little computer game 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 develop flexible and yet easy to use serious games development environments tailored for domain experts.

Much research is already underway at addressing these stated challenges, of which some have adopted Commercial-off-the-shelf (COTS) games as a potential solution. However, most COTS games available are designed specifically to entertain and some even elicit violence and sexual content, thus rendering them inappropriate (but this does not imply useless) for use in an education context [4]. Another alternative is to spearhead in-house development of serious games using open source or royalty-free game engines in collaboration with a team of developers, or ‘modding’ (modifying) COTS games by utilising a game editor application to create customised game objects and levels to suit the use of game-based learning. These approaches do not address the key challenge of facilitating the planning and development of serious games with the right mix of pedagogical, educational and fun elements. Thus, we believe there is a need to investigate the use of model-driven software engineering approaches to facilitate non-technical domain experts to plan, develop and maintain serious games for game-based learning regardless of the intricacies of the game engine/environment (platform) used.

In this paper, we present a model-driven framework designed to address the aforementioned issues. In Section II, we briefly introduce Model-Driven Engineering (MDE) and describe how such an approach can help domain experts produce serious games quickly, easily and affordably. We then review existing model-driven game development frameworks in Section III before presenting our new model-
driven serious games development framework in Section IV. Finally, we outline our further efforts in this research domain and draw conclusions on the future of this exciting field of research in Section IV.

II. MODEL DRIVEN ENGINEERING (MDE)

MDE refers to a software development approach which focuses on creation of models that represents the system-under-study (SUS) and subsequent generation of fully-working software artefacts from these models. This is made possible using a Domain-Specific Modelling Language (DSML) for describing the SUS as a model, and transformation engines and generators that bridge technology barriers in providing software solutions.

The MDE process can be defined as the following sequence of activities [5]:

i. Identifying the level of abstraction and technology platform\(^1\) to be integrated;
ii. Specifying modelling notation and abstract syntax to be used at each level of abstraction;
iii. Specifying refinement processes, and platform related information to be integrated in lower levels of abstraction;
iv. Defining generators for modelling languages used at the lowest level of abstractions (and even deployment of such code);
v. Specification of a verifier and a validator to check generated result against the upper level model, and generation of test cases for system under development.

In typical MDE practice, domain experts model the SUS using DSML to produce a domain model which is transformed into either more refined models, formatted output or software code using appropriate transformation tools. The degree of model-to-model transformation is dependent on the level of abstraction specified in the model in the defined MDE framework. In the context of game-based learning, MDE can provide the environment for domain experts to produce a serious game via modelling (either using language or visual tools) without worrying about the intricacy of game development. We firmly believe that this approach can help to simplify the route to serious games production and therefore ease the adoption of game-based learning in mainstream education and training.

MDE notably promises great benefits to its practitioners. From a software development context, MDE offers an increase in productivity, promotion of interoperability and portability among different technology platforms, support for generation of documentation, and easier software maintenance [6]. In addition, it can also lead to production of better code quality and reliability due to integration of domain rules into the DSML which minimises modelling error and increases the reliability of mapping from model to code [7]. From the non-technical domain experts’ perspective, MDE’s ability to encapsulate the technical aspects of development via a DSML massively lowers the barriers that hinder the production of applications.

III. REVIEW OF EXISTING MODEL DRIVEN GAME DEVELOPMENT FRAMEWORKS

Supporting any MDE practice is a framework that unifies models and manages the transformation between these models using specialist tools to produce the desired software artefact. The Model Driven Architecture (MDA)[8], Domain-Driven Software Development Framework [9] and Modelling Turnpike (mTurnpike) [10] are examples of model-driven frameworks that aid software architects to develop their own custom MDE solution to suit a particular domain. These model-driven frameworks have been applied in domains such as security [11], content repurposing [12], software testing [13] and pervasive computing [14].

In the field of game development, the use of software frameworks and tools are usual practice among professional (commercial) game developers. Although current tools improve the 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 generally presents no problems in the commercial computer games sector, however, in the case of game-based learning there is a greater requirement (both from a financial perspective and a skill-base level) to apply MDE. This therefore presents a practical solution to assist non-technical domain experts in the production of serious games.

At present, research in this area is still in its infancy. The SharpLudus Game Software Factory [15] is an early attempt of a model-driven approach to increase development productivity for teams developing 2D adventure games. The framework [16] consists of:

- A domain-specific modelling language - SharpLudus Game Modelling Language (SLGML) that allows the game designer to model the flow of the game using tools such as a room designer and a info display designer;
- A semantic validator that performs checks on the model to ensure the design conforms to the required semantics of SLGML; and finally,
- A Code Generator built on top of the Microsoft Visual Studio Integrated Development Environment (IDE) targeted towards generation of C# code for the associated game engine developed using DirectX technology.

Reyno & Cubel’s [17] Model-Driven Game Development (MDGD) approach introduces the use of a selection of UML diagrams to gather required information to automate generation of program code for 2D platform games. The framework is composed of:

- Two Platform Independent Models components; UML Class Diagrams extended with stereotypes are used to model the relationship between different game entities within the game world while UML State Transition Diagrams are used to model behaviour of the game entity;

\(^1\) Technology platform refers to the software framework that supports the operation of software on specific hardware. Java and the .Net Common Language Runtime (CLR) are examples of technology platforms that provide developers with Application Programming Interfaces (API) for developing software to run on a wide range of devices.
A Platform Specific Model to map game actions to hardware controls, and
A transformation tool to translate these models into C++ source code compliant with the Haaf Game Engine\(^2\).

Altunbay, Cetinkaya, & Metin’s [18] model-driven approach for developing board games shares some similarity with the MDGD approach described above. It uses UML Class Diagrams as the modelling language to represent the game model. The framework comprises of:

- A Board Game Metamodel represented with UML Class Diagrams;
- Game Domain Specific Language\(^3\) (GameDSL) - a meta-model developed specifically to aid the definition of game logic.
- A model-to-model transformation tool to transform the Board Game Meta Model onto GameDSL and model-to-text transformation tool which subsequently transform the Board Game Model in GameDSL view to compatible Java source code.

The SharpLudus Game Software Factory, MDGD and Board Game Model are research-driven attempts to adopt a model driven approach towards developing 2D games. These frameworks may appear to share a similar architecture but each differs in terms of the models used, tools available to assist the development process 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 the models and new code generation to facilitate transformation of model to code. It should also be noted that these frameworks are not designed intentionally for production of serious games by non-technical domain experts. It is our belief that a sophisticated model-driven framework designed specifically to support non-technical domain experts in production of serious games on a wide variety of deployment platforms and media is needed to drive game-based learning for mass adoption. Architecting such framework to bind both the underlying game model and specific game software model is a challenge and will require extensive research on game design, game development and a strong understanding of game software frameworks.

IV. A NEW MODEL-DRIVEN FRAMEWORK FOR SERIOUS GAMES DEVELOPMENT

In this section, we present the requirements and architectural strategies for our new model-driven serious game development framework, and discuss the challenges we face during the realisation of a model-driven approach to support non-technical domain experts in development of serious games.

A. Requirements for a Model-Driven Serious Games Development Framework

Game-based learning is still a relatively new concept. Most non-technical domain experts have very little knowledge of how games are designed and developed, and have to rely heavily on collaborative efforts with commercial game developers to produce appropriate serious games. As part of this research, the requirements for our model-driven framework are formulated based on our analysis on studies of game-based learning, serious games design, games development and MDE to provide a comprehensive and collective view [1, 19-22]. These requirements are listed as follows;

- to embed best practice for designing serious games for educational and training purposes focused on five aspects of serious games design namely game-play, game structures, game content, motivation and serious game design workflow [20];
- to provide support for a myriad of game software frameworks irrespective of the underlying model and support transparency in movement between frameworks where appropriate;
- to loosely couple artefact generation within the framework so as to provide maximum flexibility for game developers to develop code generators for different game software frameworks that comply with the framework;
- to enable integration of externally produced art assets and game functionality within the framework; and
- to the encapsulate technical aspects of game design and guide users in completing serious games design requirements formally via assistive user interfaces.

The listed requirements provide crucial pointers for architecting the ideal model-driven framework for supporting development of serious games. Only a few of these characteristics are exemplified in the SharpLudus Game Software Factory [16], MDGD framework [17] and Board Game Model [18] reviewed in Section III. The basis of a model-driven approach is to logically map the requirements to corresponding and technically elaborated semantics for composition into artefacts. As recognised in the requirements presented above, the wide variability of serious games to be generated and interoperability of serious games across different game software frameworks are important factors yet these are not adequately addressed in any of the aforementioned model-driven game frameworks.

B. Architectural Strategies for building a Model-Driven Serious Games Development Framework

A simple approach to map requirements directly to the corresponding codes, described by Kelly & Tolvanen [7] as the three-level architecture, consists of a domain modelling language, a generator and a supporting domain framework. Requirements gathered during game design are represented as a game software model and artefacts are generated for a specific game software framework using a code generator. This architecture is also seen in the MDGD framework and Board Game Model which are designed mainly for technical users. Similarly, it is used in the SharpLudus Game Software Factory which utilises a wizard-based user interface. The simplicity of this architecture tightly couples the game software model to the game software framework thereby

---

\(^2\) Read more about Haaf Game Engine from [http://hge.relishgames.com/](http://hge.relishgames.com/).

\(^3\) Read more about GameDSL from [http://gamedsl.tuxfamily.org/](http://gamedsl.tuxfamily.org/).
limiting the type of serious game that can be produced. This solution still retains all the benefits of MDE described in Section II but is less appropriate for the context of this research due to the increased technical level of modelling required and lack of interoperability.

Tackling such issues would require a model that can encapsulate the technical aspects of serious games, relates well to domain experts, and supports a higher level of abstraction. The MDA [23] presents a potential solution for architecting an interoperable model-driven serious games framework of the aforementioned requirements. Unlike the three-level architecture which consists of only a single view of the software, MDA is more elaborate in the context of model representation. The model level is made of three layers of models, namely a Computation Independent Model (CIM), a Platform Independent Model (PIM) and a Platform Specific Model (PSM) each of which offers progressively refined views with a different level of abstraction. CIM represents the logic of the SUS abstracted from the system structure, while the PIM refers to a computation-dependent model of the SUS but not tied to any hardware or software platform, whereas the PSM is computation-dependent and specific to a technology or language platform. These models are transformed from one view to another using special transformation tools before finally being generated into the software code for a targeted platform. It is obvious that MDA offers a framework setup with higher level of abstractions which allows these models (CIM and PIM) to be reused and different resultant models (PSM) to be generated via alteration of transformation rules.

From a serious games engineering perspective, the CIM represents the logical structure and building blocks of a serious game, while the PIM depicts the serious game as a software system independent of specific game technology representation. The PSM incorporates the required specific game technology details of a selected deployment platform to finalise representation of serious games. Game technology can exist in the form of game software frameworks or derived directly from multimedia APIs.

Serious games can be scripted via a scripting facility such as Lua [24], coded using programming languages such as C and C++ or made with proprietary level data for a specific game software framework. It may seem that to achieve game software variability, the transitive relationship from PIM to PSM and generation of artefacts from PSM to target can be merged giving greater flexibility for game developers to have better control over the artefacts generated by directly mapping the game software model to artefacts in whichever way seems valid. Although this approach is valid and simpler, it gives little control over the quality of serious games generated. The additional PSM provide a mechanism to maintain control over the quality of generated artefacts. Framework specific serious game software can be modelled with more detailed care and attention by the framework developer resulting in a direct one-to-one specification to code mapping thereby simplifying the generation of software code.

C. The Model-Driven Serious Games Development Framework

Our new model-driven serious game framework is featured in Figure 1 and consists of nine modules namely: (1)User Interface (UI), (2) Models, (3) MDE Tools, (4) Components Library, (5) Code Templates, (6) Artefacts, (7)Technology Platform, (8)Operating Platform and (9) Software. This configuration loosely couples the modules allowing the framework developer to flexibly substitute modules while maintaining the integrity of relationships among the modules via well-defined interfaces. It also clearly divides the views of entities while promoting structured and systematic workflow.

Encapsulating the models is the UI (1) module which can support input mechanisms such as natural language, script or even visual language, for example UML or flowchart, to specify the visual aspects of games. Separating the UI from the models enables the framework to be more accessible by different user groups, for example a wizard-based interface would suit non-technical users, natural language and visual notation might be more appropriate for intermediate users while script can be an option for advanced users. In this model-driven framework the UI module represents the modelling environment while hiding the technical details of serious games development from domain experts. We believe that separating the UI elements from model allows the framework developer to develop the right mix of assistive user-interfaces and relevant pedagogic and domain related vocabulary to guide non-technical domain experts to describe their serious games effectively.

At the core of this modelling framework is the models (2) module which represents a serious game in three different viewpoints namely:

- Game Content Model (GCM) - This represents the logical design specification of serious game as a model. The GCM will be modelled by domain experts via a modelling facility defined in the UI module. It will be linked during modelling to indicate assets and other components used in the serious game and supported by the framework. A complete GCM consists of models that represent the core aspects of serious games including definition of objects, their attributes, behaviours and linkages with art assets and game functionalities, events and progression, construction of a situation which consists of characters, objects, objectives, scripted events and problems to be solved through game-playing, tracking of interactions and various user interfaces for selection of game modes and display of information such as game objectives and results.
**Game Technology Model (GTM)** (GT) is a computation-dependent model of serious games independent of technology or language platform. The GTM, mapped from the GCM, models serious games from a software perspective representing the serious game in programmatic order and structure marked with additional and specific game-related functionality required by the serious game design. The aforementioned GameDSL is an example of GTM which can be adopted in this framework.

**Game Software Model (GSM)** refers to the transformed model of the serious game specific to a technology platform. By utilising this approach, it can be seen that a single game (GCM) can be used to support a wide variety of platform and technology without change. Models are transformed from one viewpoint to another automatically. The GCM is transformed to GTM and subsequently to GSM and finally to artefacts using appropriate MDE tools. Models described can be represented in textual form such as eXtensible Markup Language (XML), or using graphical notations such as UML. Additional information is added to the GCM during transformation to GTM and more game software framework specific information is automatically added to GSM during the transformation. Transformation can be performed using specific MDE tools (3); a transformation engine can be used to mark models with additional information and generators can be used to export models into human-readable format (such as UML diagrams) or software code. The GTM transformation engine reads the GCM and represents the game as a software model. The GTM is read by the GSM transformation engine which reorganises the games as a software model compatible with a specific game software framework by replacing game logical software constructs with the corresponding physical game software constructs. Finally, the GSM is interpreted by a generator to compose software code from predefined code templates (5) through mapping techniques. Game software framework constructs are defined by the framework developer collaboratively with developers of game software frameworks, while code templates are defined externally by framework developers by referencing the specification of GSM.

The technology platform (7) and operating platform (8) modules are developed externally by their respective parties. The configuration of this model-driven framework enables the support for a wide range of game software frameworks available in the market from open source to commercial promoting interoperability and variability of serious games.

**D. Challenges for Model-Driven Serious Games Development**

The benefits of model-driven serious games development are certainly attractive to many, but realisation of such a development approach is still challenging for developers.
These challenges include definition of modelling languages that can explicitly represent the problem domain, integration of models from multiple and overlapping views that may result in conflict, maintaining the consistency of models once transformed, testing of the transformation process and integration of generated code with foreign code. These challenges must be addressed before any model-driven serious games development environment can be developed to facilitate non-technical domain experts in development of serious games for game-based learning.

V. CONCLUSIONS

Game-based learning has great educational potential as described in [1] and is a highly desired technology-assisted learning approach for the “PlayStation-driven” generation of learners. It can help to captivate and motivate learners to learn about a topic, get them involved in solving interesting problems and is effective at unlocking their curiosity to learn more about the topic outside of the gaming session therefore promoting deep learning. However, it lacks technological solutions that can help non-technical domain experts to author customised interactive learning content to support such an innovative learning approach.

By infusing game development with practices of MDE, we believe that non-technical domain experts can produce serious games quickly, easily and affordably for use in game-based learning. This technological solution can drive the mass adoption in mainstream education and training to facilitate effective learning and better prepare young learners with the necessary knowledge and skills required of a 21st century workforce.

The model-driven framework presented in this paper serves as a reference for those who wish to develop their own model-driven serious games development environment. Using the framework, developers can implement a high-level serious games authoring environment crafted specifically for non-technical domain experts either from existing MDE technologies such as integrated modelling environments or integrating existing code frameworks.

The next step of our research is to further define our GCM, GTM and GSM. Once models are in place, we will then develop a model driven serious game development environment based on the proposed model-driven framework and conduct case studies to demonstrate applicability of model-driven engineering in assisting non-technical domain experts in developing serious games for games-based learning.

VI. REFERENCES