A Workflow for Model Driven Game Development

Conference Paper · September 2015

DOI: 10.1109/EDOC.2015.23

4 authors:

Hong Guo
Norwegian University of Science and Technology
27 PUBLICATIONS **139** CITATIONS
SEE PROFILE

Hallvard Trætteberg
Norwegian University of Science and Technology
52 PUBLICATIONS **542** CITATIONS
SEE PROFILE

Alf Inge Wang
Norwegian University of Science and Technology
150 PUBLICATIONS **2,137** CITATIONS
SEE PROFILE

Shang Gao
Örebro University
83 PUBLICATIONS **1,100** CITATIONS
SEE PROFILE

A Workflow for Model Driven Game Development

Hong Guo
School of Business Administration
Anhui University
Hefei, P.R. China
homekuo@gmail.com

Hallvard Trætteberg, Alf Inge Wang, Shang Gao
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
hal, alfw, shanggao@idi.ntnu.no

***Abstract*—Software development faces challenges from high expectation of software qualities, complexity of software and long development cycle. While Domain Specific Modeling (DSM) is helping developers overcome many of these challenges in many domains, it is not generally applied in the computer game domain. DSM can be hard to apply in the computer game domain because of the complexity of computer game domain knowledge and the peculiarity of traditional computer game development process. Without fully understanding these issues and properly solving them, the strength of DSM approaches will be constrained and game developers will be reluctant to use DSM. In this article, we investigate the development process and explore the feasibility of fitting DSM tasks in traditional computer game development in a compact way to lower cost and improve software quality. We introduce the workflow and illustrate the usage of it by presenting a case study. Further, we discuss the benefits and costs of involving DSM solutions in computer game development. Finally, we present the limitations and future work.**

Keywords-Model Driven Development, Computer Game Development, Pervasive Game, Software Engineering, Domain Specific Modeling

I. INTRODUCTION

Computer game development has been more and more challenging during the last decade due to the overall increase of project size and complexity, as well as the involvement of new technologies. Pervasive game is one emerging game genre which intensively utilizes new technologies such as GPS, sensor technologies and mobile computing. Pervasive games use new technologies to involve physical and social factors in the virtual game world, and try to make games available anywhere and anytime in our everyday life [1-3]. Such features make games more complex and the development of such games more difficult than before. Many pervasive games are built upon existing traditional games like treasure hunting games and killer games. We will use a treasure hunting game's pervasive version for the case study part in this article.

Various authoring tools [4-7] have been used to ease the game creation by providing easy user interfaces and code generation. But such tools are usually targeted at specific and simple domains. They do not address the complexity and peculiarity required by more sophisticated or innovative games. And they usually do not provide effective interfaces to extend/ customize features for such domains [8]. As a result, although they are popular, most users of such tools are beginners or amateurs instead of industrial professionals [8].

Compared with such authoring tools, traditional game engines are more powerful and flexible. They are the state of art and mainstream tools that are used by the industry to create commercial games (especially AAA title games). However, as indicated in [8], game engines also have some drawbacks. *First*, although game engines provide more freedom for customizing features, the freedom is still limited within a pre-defined game domain. Some innovative domains like pervasive games [9] and education games [10] that emerge during recent years, may not be able to benefit from existing game engines as expected. *Second*, game engines are usually huge, complex, and lack of usability. This makes the learning curve steep and the using cost rather big, even if most of the game engine features are not used.

To ease the game creation for emerging or specific domains such as the pervasive game domain, we propose an approach to create tools according to specific domain requirements, and then create games with these tools. By providing easy user interfaces and code generation similarly, creating games can also be easy, quick and with good quality. Because these tools are tailored to the target domain, learning and using them do not require more efforts than necessary.

Model Driven Software Development (MDSD) can be a good starting point of the approach. MDSD has been successfully applied in many domains to achieve similar requirements [11-15]. MDSD advocates models as the primary focus and products [16] that need to be created. And according to the models, automation techniques are used to generate codes. MDSD frees developers from writing similar codes as what software libraries do. In addition, MDSD provides more flexibility and visibility. Therefore, MDSD helps developers improve productivity and quality as well as maintainability of software systems [17]. MDSD is domain specific. This means that both code generators and modeling languages need to be domain specific [18]. Thus we use MDSD and DSM (Domain Specific Modeling) interchangeably in this article.

Some researchers presented their initial proposals and findings to apply MDSD in game development [19-24]. However, few of such application have been followed up with successful evidences of application. Both MDSD and game development are not easy. Application of MDSD in game development should consider the specialities and difficulties of both MDSD and game development. As indicated in [16], introducing new technology and techniques into an existing production environment implies not only the new software must work within legacy software, but also the development process and environment must be integrated into the legacy ones. While most researchers focused on the Domain Specific Language (DSL) construction, tools utilizing and other technical parts, it is difficult to find comprehensive studies which adapt the DSM workflow for game development. In our research, we studied the procedural characteristics of both game development and MDSD. Further, we investigated main tasks within two processes and the feasibility to connect them in an efficient way. We proposed a combined workflow that aims to provide a comfortable environment for the tasks involved in both game development and DSM. The workflow also supports task outputs be used mutually in an efficient way.

The remainder of this article is organized as follows. Section II introduces the traditional game development process as well as a typical model driven development process. Section III elaborates our proposed workflow. Section IV presents a case study to demonstrate the usage of workflow for developing a pervasive game DSM solution. Later, Section V discusses gains and costs of involving DSM in our workflow, limitations of our work, as well as future works. We conclude the article in Section VI.

II. Background

In this section, we introduce background knowledge regarding traditional computer game development process and model driven software development process.

A. *Game development workflow*

Game development is usually carried out in four stages: Conception, Pre-production, Development, and Validation [25]. While Conception stage initiates basic features and rough plans, Validation stage tests game products, Pre-production and Development are the stages where the majority of game product is produced. Within Pre-production and Development stages, design and implementation of software programs and other artefacts are generated. Therefore, these two stages are also named as Elaboration stage by some researchers [26]. They are also the subjects of our research where we introduce DSM to improve game software development. In this article, "game development" means game development within these two stages instead of the four stages. We will introduce the main tasks that are carried out within these two stages relating to software development in the following part of this section.

As presented in Fig. 1, the Pre-production stage is mainly devoted to the game design and a game prototype. *Game Design* describes the definition of the main aspects of a game world like epoch, style, context, goals, types of objects, and user perceptions [25]. The prototype helps participants understand what the core mechanic is and how activities in the game become meaningful over time. We call this prototype *Baseline Prototype* since it is the first working version of the formal system, and needs to be distinguished with other prototypes in the development stage.

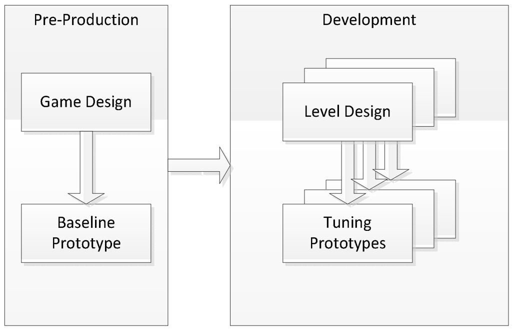

Figure 1. Traditional computer game development workflow.

The Development stage is the stage where game elements are created and integrated. Similar to tasks in the Pre-production stage, these tasks involve both design and programming. However, the focus differs. The design in the pre-production stage focuses on high-level and common features like global challenges, actions, and types of world elements. While the design in the development stage elaborates the game design and focuses on more specific challenges, actions, objects of world elements and various ways to integrate them into game levels. This is also true for games that have only one level. Thus it is usually called *Level Design*. To ensure the design quality, a lot of iterations are involved in this stage. In each iteration, prototypes are created, evaluated and enhanced to consolidate the level design. This is mainly because unless you can really play the game, it is always difficult to say whether a gameplay is playable or not [26]. Also, the more complex a level is, the more difficulty imagining and judging the gameplay is. Thus levels are generally developed with prototypes of increasing complexity [25]. We call these prototypes as *Tuning Prototypes* as they are mainly used for gameplay tuning and game elements integration.

B. *Domain specific modeling workflow*

DSM application involves two parts: Definition of DSM and Usage of DSM. *Definition of DSM* defines domain specific languages and develops corresponding tools such as DSL editors, Domain Specific (DS) Libraries and code generators. *Usage of DSM* specifies models within DSL tools and generates codes with these models. DSL development is difficult since it requires both language development expertise and domain knowledge. To define a DSL, domain analysis is used to collect and analyze domain knowledge, and then output results in a preferable format. Usually there are four deliverables that are produced from domain analysis: domain definition, domain terminology, domain concepts descriptions, commonality and variability descriptions as well as their interdependencies [27]. These domain analysis outputs contribute to DSM artefacts in different ways. Among them, Commonality and Variability analysis are especially important. *Commonality* analysis identifies useful abstractions and can be used to define the primitives of the language as well as the common execution patterns. Therefore it often contributes a lot to the domain specific libraries' construction. *Variability* analysis indicates the potential changes among all the models written in the expected DSL. And it mainly devotes to the DSL meta-model's construction [27]. With the meta-model and *Other Inputs* such as syntax definition and semantics definition, the DSL is able to be defined. Further, a DSL editor can be constructed by utilizing tools. A generator can be defined afterwards to make code automation possible. It is practically efficient and recommended to use *Reference Codes* from working prototypes to define the generator templates as a start [18].

Compared with DSM Definition, DSM Usage is simpler. With the support from various language workbenches ([28], [29]), *Codes* can be automatically generated by generators from *Models* that are written in the corresponding DSL. The inputs that are needed to define a DSM solution are presented in Fig. 2, and the inputs and outputs to use the DSM solution as well.

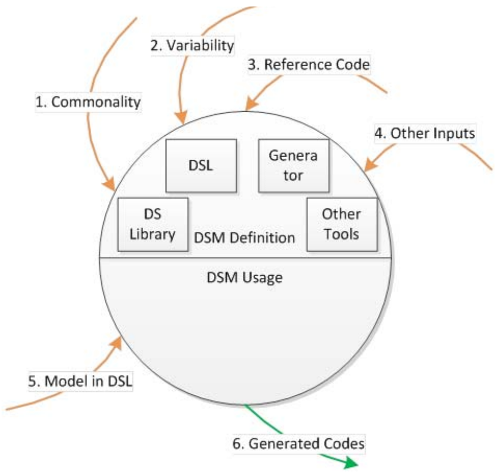

Figure 2. Domain specific modeling inputs and outputs.

C. *State of Art Model Driven Game Development Process*

Compared with traditional business software development, game software development is more difficult [30] and special as introduced previously. As pointed out in [16], a prudent and practical way to introduce new techniques into an existing production environment implies that the new process and environment should be integrated into the legacy one in addition to the technologies. The process of model driven game development must address the original procedural specialities of computer game development before practitioners would use it and benefit from it. However, to our best knowledge, few articles have addressed the procedural issues when applying MDSD approaches to game development. We examined the following aspects of such practices:

1. Whether a general process was proposed;
2. Whether the process supports iterative evolution;
3. Whether the process was an adapted DSM process with consideration about game development peculiarities.

The review result is summarized in TABLE I. In the table, "*"s indicate that the research in current row meets the criteria in current column. Although our research focus was language (DSL) oriented approach, we also reviewed some other model driven approaches. For instance, MDSD was considered to be an effective way to implement Software Product Lines (SPL) [31], but using MDSD to implement SPLs does not have to involve DSLs. [32] and [33] are examples of such Software Product Line (SPL) approach.

Among the articles we have reviewed, many DSL practices were introduced without mentioning the process issues ([22, 23, 34-36]). Only six of the articles illustrated the overall process with figures and explanatory texts. As the table presents, Zhang [33] used a meta programming language (and tool & method) named XVCL [37] to implement a software product line. The process introduced was specialized for XVCL but not for game development. Also, the process was based on XVCL environment instead of a general one. Nascimento et al. [32] used product line approach as well. The authors identified three distinct phases in the overall process: Component Modeling, Component Implementation, and Component Testing. For each phase, control, input, output, and mechanism were specified. It can be thought as some kind of adaptation for the process, but lacked detailed information about how to transform inputs to outputs. Also, it was not a language-oriented MDSD approach.

TABLE I. STATE OF ART MODEL DRIVEN GAME DEVELOPMENT PROCESS

Literature	General	Iterative	Adapted	DSL
[38]	*	*	*	*
[24]	*	*	NO	*
[39]	*	*	NO	*
[28]	*	NO	NO	*
[32]	*	NO	*	NO
[33]	XVCL based	NO	NO	NO
[22, 23, 34-36]	NA			*
[35]	NA			NO

In [39], the authors illustrated their approach from three views: products and activities view, sequencing view, and participants and roles view. An incremental approach was adopted where both the language and its implementation would be able to evolve (supporting iterative evolution). This was because "this evolutive nature alleviates the costs" and "these activities can be interleaved with the production of videogames". Such a pragmatic posture was very close to ours but the process was not illustrated with details on how to adapt traditional game activities for DSM usage. Similarly, in [24], the authors suggested that the language engineering process was part of the game design phase. This means that in an iterative process where the game design was refined stepwise, the used DSLs could be refined as well to fit the new requirements best. However, the authors did not address the issue on how to bridge the tasks within game production and DSM with more details.

In [28], the authors proposed to use language workbenches to implement (DSLs and) software product lines for game development. The approach illustrated roles, artefacts and workflows from the perspective of general (model driven) software development. But the approach did not temp to make adaptation for the specific game domain. Furtado et al. [38] introduced about utilizing DSLs and software product lines to develop computer games also. The authors emphasized the iterative way to evolve the main artefacts (feature model, DSL, generator, and reference samples). Further, more detailed steps about envisioning the scope, analyzing the domain, assessing automation potentials, and creating application &development assets were also proposed. Such steps were illustrated with some details regarding game domain. The motivation was very close to ours, but lacked adequate technical details to regulate the transformation from game artefacts to DSM artefacts.

III. A DSM EMBEDDED WORKFLOW FOR GAME DEVELOPMENT

In this research, we investigated tasks and artefacts within normal DSM approach and traditional computer game development. Further, we proposed connections between them to minimize efforts of involving DSM. The main connections include:

1. Traditional game design and (part of) level design are regulated to produce domain analysis outputs in an efficient way;
2. Traditional baseline prototype is reused to construct code generator;
3. Instead of manually programming, tuning prototypes are fully generated from a game specification given in the DSM editor.

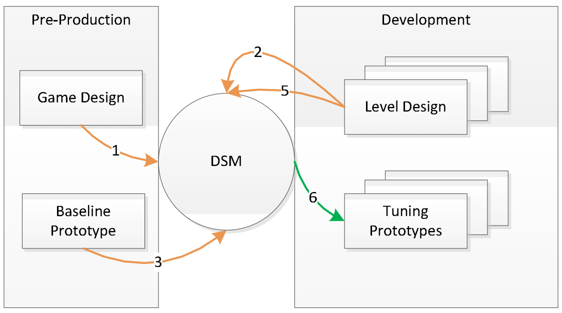

Figure 3. A DSM embedded workflow for game development.

The overall workflow is presented as Fig. 3. From the figure, we can see that each main task in game development is attached with one or more input/outputs of the DSM. This indicates a connection between game tasks and DSM tasks. Other inputs (as shown as the arrow line numbered 4 in Fig. 2) do not appear in this figure because they are not related to game development tasks. Instead, they are related to some subjective elements like DSL syntax definition. By applying such a workflow, we keep respecting the special characteristics of game software development. In addition, we make use of existed game tasks to produce important inputs to the DSM definition. Then, we benefit from codes automation for numerous prototypes. We will explain the connections in the following sub-sections. Before that, one of our research results - Pervasive Game Ontology (PerGO) is briefly introduced. PerGO is important to our workflow because it regulates game design and level design to produce important DSM artefacts, and therefore plays an important role to the overall efficiency and quality of DSM involvement.

A. *Pervasive game ontology*

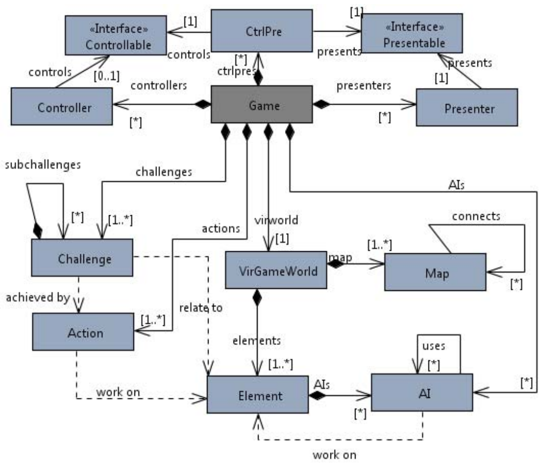

Figure 4. Core part of PerGO.

PerGO [40] is a domain vocabulary for the pervasive game domain. PerGO contains around 100 commonly used concepts. These concepts are classified in six perspectives: Gameplay, AI, Virtual Game World, Control, Presentation, and CtrlPresentation. In contrast to some other existing ontologies or frameworks within computer game domains (e.g., [41]), PerGO focuses on identifying concepts that are promising to work as concepts in a DSL for a specific pervasive game domain. Three criteria are used to make or identify such abstractions: 1. the concepts should be of proper complexity. Simple attributes that can be expressed to variables of primitive type are thought to be of improper complexity and are excluded; 2. The concepts should be of proper abstraction level. Abstract concepts which are difficult to implement as a data structure and concrete concepts which do not appear in the common discussion about design are thought to be of improper abstraction level. This is used to balance between the expressiveness and the flexibility of the DSL. 3. The concepts should be constructive. In another word, the concepts describe some nouns, like some thing or some rule that can be constructed. They should not be some adjectives, like 'interesting' or 'intuitive'.

The idea for creating such an ontology is to structure domain analysis work, and provide semi-finished products (concepts and relationships) to ease the DSL construction. Compared with other classic and general domain analysis approaches (e.g., FODA [42]), using a pre-defined domain vocabulary enables better knowledge reuse. Therefore, the domain analysis can be more qualified and efficient. We will illustrate how domain analysis work is structured by utilizing PerGO in the following two sections.

PerGO can be divided into two parts: a core part which contains higher level abstractions that are common to computer games, and a pervasive game part which contains lower level concepts that are primarily or often used by pervasive games. Fig. 4 shows the core part of the ontology, and more information of the pervasive part can be found in [40].

B. Game design vs. commonality analysis

Game design includes global settings like epoch, style, historical or mythical references, and high-level organization like level hierarchy, navigation graph, or topology. It also includes some common elements that can be used to construct levels like main characters, classes of game objects, image and sound chart. Traditional approaches use a game design document to describe game design. In that document, detailed game design is fully described mainly in texts or drawings with commented texts. Some diagrammatic presentation can also be utilized. For instances, flowchart for menu navigation or level navigation may be drawn as standalone documents or attachments to the game design document [43]. In our workflow, we encourage that this common part of design can be expressed in a more structured way, so that important information can be reused for DSM in a more efficient way. We propose the following table (TABLE II) to be filled in to highlight and organize the commonalities. When filling in the table, game designers should refer to PerGO, and go through game design according to the six perspectives. They should decide which concepts should be used in each perspective, and decide more common information for each concept. These concepts are parts of the commonality. In addition, they are also the base to discuss about more detailed commonality and variability. PerGO is designed to provide plenty of concepts for pervasive game development. But for some specific pervasive games, some concepts may not be covered by PerGO (like Treasure in treasure-hunting games). In that case, designers need to add these specific concepts in the table. This action will not bring much impact to the DSL definition and the overall workflow.

C. Level design vs. variability analysis

In one game level, designers need to define space geometry, choose positions and actions associated with characters and objects in the level, and specify goals for the level. What is more, designers need to design user interfaces to ensure all necessary information is presented to players, and players are able to interact with the game system in an expected way. These tasks are the same for a game which has only one level. Thus, to make our discussion easy and precise, we only consider one-level games in this section. We will talk about multi-level games in the discussion part. In our workflow, we propose to use level design in two ways. *First*, all the design should be imagined and presented as variability which contributes to the DSL definition. Variability should be filled in a table also (TABLE III). And s*econd*, one specific design that corresponds to one tuning prototype, should be modeled in the DSL. This model will be used to produce prototype codes when the DSM is defined and ready to be used (see Sub-section F).

As presented in [18], there are three places for a language to express variation: 1) language concepts, 2) concept attributes, and 3) concept relationships. Therefore, in our workflow, we recommend designers to go through the perspectives one more time. Also by referring to PerGO, they check whether some new concepts are needed in addition to those listed in the commonality table, and then fill them in the Concept column. Further, they imagine and scope the differences among possible designs, and fill in Variability Details column with attributes or relationships that can support such differences (variability). Similar to the commonality analysis, we assume that PerGO provides plenty of concepts for designers to use. However, it is also normal that some game specific concepts are not included. In that case, designers make corresponding abstractions and fill them in the table.

TABLE II. COMMONALITY TABLE

Perspective	Concept	Commonality Details

TABLE III. VARIABILITY TABLE

Perspective	Concept	Variability Details

D. Baseline prototype vs. reference codes

A Baseline prototype should help game developers understand core mechanics and how actions of players can achieve game goals over time. The prototype should also base on the real working environment and platform. As said in [44], it is "a working version of the formal system". Therefore, baseline prototypes contribute to the construction of code generators with working reference codes. Physical prototypes or quick prototypes that utilize ready-to-use software other than real working environment are not in our discussion scope. Pervasive game designers may choose to validate core game design by such kinds of prototypes [45]. It means that they may skip the baseline prototype construction and move on quickly to level design and tuning prototypes. In that case, reference codes will not be available until the first tuning prototype is manually written.

As indicated in [16], “many software practitioners, when first faced with the notation of MDSD, express concern about the technical difficulties involved in translating models into code”. They may have concern about whether codes will run fast enough and etc. By using a manually written and workable prototype as the source of reference codes, we can not only save time to construct the code generator, but also ensure the quality of the generated codes.

E. Level design vs. model in DSL

Ideally, when commonality analysis, variability analysis, and baseline prototype construction have been finished, it is expected that an initial DSL as well as corresponding tool chains (DSL editor, DS libraries, generators, and etc.) can be built and are available to use. At that time, level design can be specified in the DSL editor by creating a model that is written in the DSL. If some design is not possible to be expressed in the DSL, it implies some issues existing in commonality or variability analysis. Therefore, the DSL may need to be re-constructed. This is normal for DSM practices. It is not possible that one DSL can be designed once and used forever, and "a large part of its power comes from its ability to evolve" [18]. Fortunately, the game development is a highly iterative process by itself. While we expect DSL brings benefits to game development, game development provides an agile and iterative environment for DSL to evolve.

F. Tuning prototypes vs. generated codes

As mentioned previously, if DSL and related tools grow to be mature and work as expected, programmers will be able to get free from hand-writing similar tuning prototypes. At that time, these prototypes can be automatically generated from DSL models that represent the design of the prototypes. However, if the generator needs to be reconstructed or polished, part of the prototypes will need to be written manually. The new manually written codes will contribute to the generator's re-construction in the next iteration. Further, if physical prototypes are chose to replace the baseline prototype in pre-production stage, the first tuning prototype will need to be hand-written and contribute to the generator definition. But the other tuning prototypes can be generated as usual afterwards.

IV. A CASE STUDY

We will demonstrate the usage of the workflow with a pervasive game case. Below is a brief description of the game called "RealCoins".

RealCoins is a location-based, mobile version of traditional treasure-hunting games. Several groups of players can participate with mobile devices like tablet PCs or smart phones. The mobile devices need to be equipped with GPS so that position information of players can be sensed and known by the game. To play the game, all players should be physically at the same place and login to the web based game with their group IDs and player IDs. When the game starts, some hidden virtual coins inside several treasure zones and some other virtual coins outside are scattered in the game area. In the main view of the game, a real map with information of the treasure zones, coins, players are presented. Players are not able to see the hidden virtual coins before they or their group members enter the treasure zone where hidden coins locate. The main game play is that players need to move physically to enter a treasure zone (by locating within the zone) or get a virtual coin (by co-locating with the coin). An optional gameplay is to steal coins from other group players by approaching them and pressing a hot key. The action of stealing always costs a fix amount of coins of the player, while the mount of coins that the player actually steals is randomly calculated. Thus it is possible to lose some coins as a result. When the game ends after some time, the group or player with the most coins wins the game.

TABLE IV. COMMONALITY FOR REALCOINS

Perspective	Concept	Commonality Details
…	…	…
Gameplay	(Action) Move	1. The action should be invoked by physical moving. 2. The positions should be updated regularly according to GPS information.
	(Action) GetTreasure	1. The coins disappear after the action. 2. The value of coins is added to the player's wallet.
Virtual Game World	Game	1. The name of the game. 2. The introduction the game.
…	…	…

A. DSM analysis

We go through commonality and variability analysis introduced in Section III. Due to the limitation of space, only part of the result is presented here. In TABLE IV, two common concepts in Gameplay perspective and some common logic (which is expected to be implemented within functions) were recorded. Besides that, the "Game" concept in Virtual Game World perspective was also listed. Name and introduction for the game (which can be implemented as attributes) were listed in the table as well. Variability analysis was based on the result of commonality analysis. In TABLE V, it was showed that one more concept StealCoin was identified since it was an optional action (a variable in the game design space). For the other three concepts, many attributes that may vary among different game instances were also identified such as hotkeys and game duration.

B. DSM definition

After all the variability information has been collected, we organized them to construct an initial meta-model (as shown in Fig. 5.) of the DSL. In the meta-model, the concepts were directly from the variability analysis table, while the attributes were decided according to the Variability Details in the table. The relationship among them was mainly derived from PerGO, but was simplified according to the actual needs. Then, a DSL editor was generated in a language workbench tool.

TABLE V. VARIABILITY FOR REALCOINS

Perspective	Concept	Variability Details
…	…	…
Gameplay	Move	NA
	GetTreasure	1. The hot key to invoke the action. 2. The distance within which treasure can be got.
	(Action) StealCoin	1. Whether to support the action. 2. The distance within which coins can be stolen. 3. The cost of stealing. 4. Threshold value for the random result. 5. The hotkey to invoke the action.
Virtual Game World	Game	1. Whether the single player or group with most coins win the game. 2. Game duration.
…	…	…

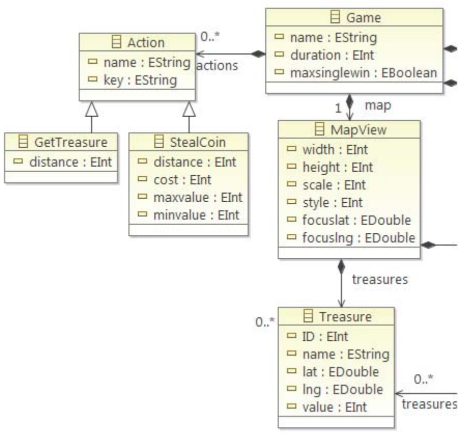

Figure 5. DSL Meta-model of Real Coins.

The commonality analysis result could contribute to the domain specific library construction. As our case was not quite complex, the commonality analysis result mainly contributed to data structures and procedural patterns in the final codes. These parts could constitute a domain specific library elsewhere if necessary.

To validate the design and technical solutions, a prototype was created. The prototype contributed a lot to the generator construction. We filled the overall workable codes to the code generation template first. Then we modified places where codes should be generated according to different game models. At this time, we have finished all the tasks for DSM definition and the DSM environment was ready to use.

C. DSM usage

When more prototypes were needed to tune gameplay or other game configurations, we only need to create a model in the DSL editor and generate the prototype codes automatically. Although we also met some circumstances where reconstruction of the DSL or the generator was necessary, the overall workload was reduced a lot. And more importantly, the production of tuning prototypes became almost bug free as the generator have evolved to be mature after several iterations. In a traditional computer game development process without DSM involvement, such bugs can often appear due to missing some modification within similar programs. According to our experience in this case, creating a new game prototype using the RealCoins DSL required writing 60 lines of code and it costed around 0.5 hours. As a comparison, the manually written baseline prototype with 1263 lines of code costed us 14 hours. Although we have invested 11 hours to construct the DSM environment, the 11 hours was a one-off cost and we could benefit for all subsequent prototypes.

D. Result and User Attitude

More details about this case have been reported in [46]. Furthermore, according to a recent user attitude evaluation on the game development approach where the present workflow was employed [47], most potential users thought the approach was useful and easy to use.

V. Discussion

In this article, we presented our workflow to make computer game development benefits from MDSD. We will discuss the value added to traditional computer game, as well as the cost by introducing DSM. We will also discuss the limitation of our workflow and possible future works.

A. Enhance the computer game development

In [48], the authors indicated that the software engineering process was not clearly understood and hindered the reliable development in computer game field. By investigating factors that lead to success or failure of computer game development, issues with transition from pre-production to development (called as production by the authors) stages were highlighted. These issues include: 1) How to transform the form of pre-production documents so that they can be used in development; 2) How to identify implied information; and 3) How to apply domain knowledge for the creative process. As a result, the authors suggested that traditional requirements engineering techniques should be improved to support creative computer game development better. We explain how our approach helps us solve these issues here. In our workflow, there are one design document (game design in the form of commonality analysis) in pre-production and two design documents (level design in the form of variability and a set of models in DSL) in development. We structuralize commonality analysis according to PerGO, and basing on the result, we perform variability analysis. Therefore, the pre-production design has been used directly and efficiently in production stage. Both commonality and variability are in a form that is very easy to be mapped to the meta-model elements of the DSL. For the detailed level design, since it is written in the DSL editor, the actual data must conform to the DSL syntax and semantic constrains. The game design in pre-production is inherited naturally to the development stage. For the second issue, we suppose by trying to detail the domain knowledge in the same framework, implied information can be identified more easily. The third issue is about how to promote the creative work with effective domain knowledge reuse. In our workflow, domain knowledge has been captured and encapsulated in the DSM artefacts (DSL, libraries, and generators), so developers can focus on the creative work. Such creative work may include instantiating gameplay and integrating various game objects. They use abstractions in the problem domain provided by the DSL editor, do their creative work, and generate codes in the solution domain.

The success of a game depends to a large extend on whether gameplays provided by the game are really appeal. But gameplays are usually hard to be judged until you can really try it. This makes the design tightly connected with the software development. In addition, the overall game development process is often highly iterative. Introducing MDSD in this process helps us reuse previous design, domain knowledge, as well as codes. Therefore, MDSD can help us save a lot of human power and development time. From the perspective of MDSD, researchers also emphasized the ability for a language to evolve [18] that may highly relies on an agile application lifecycle. In our workflow, by organically integrating DSM tasks and game development tasks, the overall process is able to keep iterative in an efficient way. In traditional game development process, game design and prototypes are refined iteratively. In our workflow, game design, DSM, and prototypes are refined iteratively. In another words, DSM accelerates the prototypes' development, while prototypes and game design provide feedback to DSM.

B. *Minimize the DSM efforts*

Involving DSM is not cheap. DSM must pay off to be successful. We tried our best to minimize the extra efforts brought by involving DSM in our workflow. We discuss about these efforts according to the constitution of DSM. There are mainly three types of artefacts that need to be developed when DSM is involved: DSL editors, domain specific libraries, and generators. *Firstly*, because concrete syntax of DSL (especially textual concrete syntax) can be generated by some tools, the main efforts that are required to develop a DSL editor are defining meta-models. The most difficult part of defining meta-models is making abstraction and analyzing variability. In our workflow, we reuse the existed game design task, and structure it by a pre-defined but extensible framework. This does not require much extra effort according to our experiences in the case study. What is more, for some simple games, the game design can probably be done in this form only (without traditional descriptions in natural language and figures). *Secondly*, regarding the domain specific libraries, the efforts needed depend on heavily on the abstraction and the implementation. While efforts to make abstractions can be lowered by commonality analysis, implementation efforts can be lowered by utilizing prototype codes. *Lastly*, the generator reuses the prototype codes directly, especially the first time when it is constructed. At that time, language developers are recommended to paste workable code snippets to the generator templates, and then replace the variable parts and refine [18].

C. *Threats to validity*

Some threats to the validity of our case study can be identified. The concepts internal validity, external validity and construct validity, defined in [49], are used here as the basis of the discussion.

Construct validity is concerned with the relationship between theory and observations. To test the applicability of the DSM embedded workflow, we applied this method in a game called 'Real Coins'. This case study was performed by ourselves. We did not find any significant problems when performing this case study. However, we noticed that it is hard to measure the difficulty other participants might have in performing the case study by using the workflow. Further, there are other possible methods to evaluate the modeling constructs in this workflow method. We plan to carry out another round of evaluation on this in the near future.

Internal validity is concerned with the relationship between treatment and outcome. In our case study, we spent around three days to finish all the tasks in the workflow method for the game proposed in the case study. The whole process was pretty straightforward. But we had not made any tutorial for others to use this workflow method. Therefore, it is quite difficult to judge how quickly people can have a good understanding of this workflow method. We also plan to make a tutorial on this soon.

External Validity is concerned with the question as to whether it is possible to generalize from the case settings to other situations. As our study was performed by users who had an educational background in information systems modeling, it is possible that the results may not generalize to other users with different modeling experience. Further, as the workflow method so far was only evaluated in limited cases, it is an open question whether the findings are applicable to other game developments.

D. *Future work*

In this article, we discussed about the possibility and feasibility to reuse existed game development tasks as inputs of DSM, and then benefit from DSM techniques with minimized efforts. Despite other potential benefits, the most direct and important benefit is that the efforts to program numerous prototypes can be saved by automation. We are discussing the tuning prototypes in this article. However, we realize that this workflow may also work for other cases. For instances, it may also work for different levels or different games in one game family/ product line. The commonality and variability among tuning prototypes, among levels, or among game family/product line members can be similar. But the emphasis may be different. This may not be a problem since we always carry variability analysis in the workflow. When DSL is used to develop different levels, the overall DSM solution can be thought as a systematic and efficient way to develop level editor for one specific game domain. Such a level editor is usually available together with game engine and needs to be customized by handwriting.

Since the efforts for variability analysis depend heavily on the domain complexity and the quality of the pre-defined ontology (e.g., PerGO), extending and refining such domain vocabularies is important to us in the future. Other future work includes the research on other aspects of the proposed approach (organizational aspect for instance), development of an IDE and benchmark applications based on current approach, and further evaluation of the proposed approach based on them.

VI. CONCLUSION

Game development can be special since the game design is usually tightly associated with the software development and the overall design and software development involves numerous prototypes. Applying MDSD in games development should not be a one-side research. Game development peculiarities should be carefully investigated and be handled in the MDSD process. In this article, we proposed a workflow to fit DSM tasks in traditional game development process to lower cost and support evolution of both game design and software development. We illustrated this process by a case study and discussed benefits and costs of involving such a process. According to the results from the case study, we found that the applicability of the process is quite good.

REFERENCES

[1] M. Montola, J. Stenros, and A. Waern, Pervasive games: theory and design: Taylor & Francis US, 2009.

[2] E. Nieuwdorp, "The pervasive discourse: an analysis," Computers in Entertainment (CIE), vol. 5, 2007.

[3] C. Magerkurth, A. Cheok, R. Mandryk, and T. Nilsen, "Pervasive games: bringing computer entertainment back to the real world," Computers in Entertainment (CIE), vol. 3, pp. 4-4, 2005.

[4] E. J. Marchiori, J. Torrente, Á. del Blanco, P. Moreno-Ger, P. Sancho, and B. Fernández-Manjón, "A narrative metaphor to facilitate educational game authoring," Computers & Education, vol. 58, pp. 590-599, 2012.

[5] Y. Wang, T. Langlotz, M. Billinghurst, and T. Bell, "An authoring tool for mobile phone AR environments," in Proceedings of New Zealand Computer Science Research Student Conference, 2009, pp. 1-4.

[6] M. Gandy and B. MacIntyre, "Designer's augmented reality toolkit, ten years later: implications for new media authoring tools," in Proceedings of the 27th annual ACM symposium on User interface software and technology, 2014, pp. 627-636.

[7] R. Wetzel, L. Blum, and L. Oppermann, "Tidy city: a location-based game supported by in-situ and web-based authoring tools to enable user-created content," in Proceedings of the International Conference on the Foundations of Digital Games, 2012, pp. 238-241.

[8] A. W. Furtado and A. L. Santos, "Using domain-specific modeling towards computer games development industrialization," in The 6th OOPSLA Workshop on Domain-Specific Modeling (DSM06), 2006.

[9] H. Guo, H. Trætteberg, A. I. Wang, and M. Zhu, "TeMPS: A Conceptual Framework for Pervasive and Social Games," in 2010 IEEE 3rd International Conference on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL 2010), 2010, pp. 31-37.

[10] K. L. McClarty, A. Orr, P. M. Frey, R. P. Dolan, V. Vassileva, and A. McVay, "A literature review of gaming in education," Gaming In Education, 2012.

[11] B. Hailpern and P. Tarr, "Model-driven development: The good, the bad, and the ugly," IBM Systems Journal, vol. 45, pp. 451-461, 2006.

[12] J. Hutchinson, J. Whittle, and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," Science of Computer Programming, vol. 89, pp. 144-161, 2014.

[13] H. Trætteberg, "Model-based user interface design," 2002.

[14] J. M. Gascueña, E. Navarro, and A. Fernández-Caballero, "Model-driven engineering techniques for the development of multi-agent systems," Engineering Applications of Artificial Intelligence, vol. 25, pp. 159-173, 2012.

[15] A. Leitner, C. Preschern, and C. Kreiner, "Effective development of automation systems through domain-specific modeling in a small enterprise context," Software & Systems Modeling, vol. 13, pp. 35-54, 2014.

[16] B. Selic, "The pragmatics of model-driven development," Software, IEEE, vol. 20, pp. 19-25, 2003.

[17] T. Stahl, M. V lter, and K. Czarnecki, Model-driven software development: technology, engineering, management: John Wiley & Sons, 2006.

[18] S. Kelly and J.-P. Tolvanen, Domain-Specific Modeling Enabling Full Code Generation: John Wiley & Sons, Inc., 2008.

[19] A. W. B. Furtado, A. L. M. Santos, and G. L. Ramalho, "SharpLudus revisited: from ad hoc and monolithic digital game DSLs to effectively customized DSM approaches," presented at the Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11, Portland, Oregon, USA, 2011.

[20] P. Moreno-Ger, J. L. Sierra, I. Martínez-Ortiz, and B. Fernández-Manjón, "A documental approach to adventure game development," Science of Computer Programming, vol. 67, pp. 3-31, 2007.

[21] R. Albright, A. Demers, J. Gehrke, N. Gupta, H. Lee, R. Keilty, et al., "SGL: a scalable language for data-driven games," in Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, pp. 1217-1222.

[22] F. E. Hernandez and F. R. Ortega, "Eberos GML2D: a graphical domain-specific language for modeling 2D video games," presented at the Proceedings of the 10th Workshop on Domain-Specific Modeling, Reno, Nevada, 2010.

[23] H. Behrens, "Mdsd for the iphone: developing a domain-specific language and ide tooling to produce real world applications for mobile devices," in Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, 2010, pp. 123-128.

[24] R. Walter and M. Masuch, "How to integrate domain-specific languages into the game development process," presented at the Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology, Lisbon, Portugal, 2011.

[25] V. Gal, C. Le Prado, S. Natkin, and L. Vega, "Writing for video games," Proceedings Laval Virtual (IVRC), 2002.

[26] E. Adams, Fundamentals of game design: New Riders, 2010.

[27] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," ACM Comput. Surv., vol. 37, pp. 316-344, 2005.

[28] S. Maier and D. Volk, "Facilitating language-oriented game development by the help of language workbenches," presented at the Proceedings of the 2008 Conference on Future Play: Research, Play, Share, Toronto, Ontario, Canada, 2008.

[29] M. Fowler, "Language workbenches: The killer-app for domain specific languages," 2005.

[30] B. Jonathan, "Game Development: Harder Than You Think," Queue, vol. 1, pp. 28-37, 2004.

[31] K. Czarnecki, M. Antkiewicz, C. H. P. Kim, S. Lau, and K. Pietroszek, "Model-driven software product lines," in Companion to the 20th annual ACM SIGPLAN conference on Object-oriented

programming, systems, languages, and applications, 2005, pp. 126-127.

[32] L. M. Nascimento, E. S. de Almeida, and S. R. de Lemos Meira, "A case study in software product lines-the case of the mobile game domain," in Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference, 2008, pp. 43-50.

[33] W. Zhang, "Architecturally reconfigurable development of mobile games," in Embedded Software and Systems, 2005. Second International Conference on, 2005, p. 7 pp.

[34] S. Tang, M. Hanneghan, T. Hughes, C. Dennett, S. Cooper, M. A. Sabri, et al., "Towards a Domain Specific Modelling Language for Serious Game Design," in 6th International Game Design and Technology Workshop (GDTW'08), Liverpool, UK, 2008, pp. 43-52.

[35] E. M. Reyno and J. Á. Carsí Cubel, "Automatic prototyping in model-driven game development," Computers in Entertainment (CIE), vol. 7, p. 29, 2009.

[36] M. Funk and M. Rauterberg, "PULP scription: a DSL for mobile HTML5 game applications," in Entertainment Computing-ICEC 2012, ed: Springer, 2012, pp. 504-510.

[37] XVCL, "http://sourceforge.net/projects/fxvcl/."

[38] A. W. B. Furtado, A. L. M. Santos, G. L. Ramalho, and E. S. de Almeida, "Improving Digital Game Development with Software Product Lines," Software, IEEE, vol. 28, pp. 30-37, 2011.

[39] P. Moreno-Ger, I. Martínez-Ortiz, J. L. Sierra, and B. F. Manjón, "Language-driven development of videogames: The< e-Game> experience," in Entertainment Computing-ICEC 2006, ed: Springer, 2006, pp. 153-164.

[40] H. Guo, H. Trætteberg, A. I. Wang, and S. Gao, "PerGO: An Ontology Towards Model Driven Pervasive Game Development," in Ontologies, DataBases, and Applications of Semantics, Amantea, Italy, 2014.

[41] M. M. Jose Zagal, Clara Fernandez, Juan Pablo Ordóñez, "Game Ontology Project," 2011.

[42] S. G. C. K C Kang, J A Hess, W E Novak, A S Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," November 1990.

[43] R. Rouse III, Game design: Theory and practice: Jones & Bartlett Learning, 2010.

[44] T. Fullerton, C. Sawain, and S. S. Hoffman, Game design workshop: designing, prototyping and playtesting games: Taylor & Francis US, 2004.

[45] E. M. Ollila, R. Suomela, and J. Holopainen, "Using prototypes in early pervasive game development," Computers in Entertainment (CIE), vol. 6, p. 17, 2008.

[46] H. Guo, H. Trætteberg, A. I. Wang, and S. Gao, "RealCoins: A Case Study of Enhanced Model Driven Development for Pervasive Game," International Journal of Multimedia and Ubiquitous Engineering, vol. 10, 2015.

[47] H. Guo, S. Gao, J. Krogstie, and H. Trætteberg, "An Evaluation of an Enhanced Model Driven Approach for Computer Game Creation," in Enterprise, Business-Process and Information Systems Modeling, ed: Springer, 2015.

[48] D. Callele, E. Neufeld, and K. Schneider, "Requirements engineering and the creative process in the video game industry," in Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on, 2005, pp. 240-250.

[49] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in software engineering: Springer, 2012.