Lessons from Practicing an Adapted Model Driven Approach in Game Development

Hong Guo, Hallvard Trætteberg, Alf Inge Wang, Shang Gao,
and Maria Letizia Jaccheri

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
{Guohong,hal,alfw,shanggao,letizia.jaccheri}@idi.ntnu.no

Abstract. Various authoring tools have been used to ease the game creation. However, these pre-defined tools may not be suitable for some emerging or special domains. We proposed an approach named Game Creation with Customized Tools (GCCT) to create tools for certain domains first, and then create games using these tools. GCCT is based on the widely applied Model Driven Development (MDD) approach. Despite the apparent appropriateness and benefits, MDD also has drawbacks. Among them, non-trivial cost for tools development is prominent. To address this, some enhancements were made in GCCT, and two case studies were performed to evaluate the cost and the productivity when involving GCCT. In this paper, we reported the results of the case studies as well as practical lessons we have learnt.

Keywords: Computer Game Development, Model Driven Development, Cost.

1 Introduction

Various authoring tools have been adopted to ease computer games creation by providing easy user interfaces and code automation. However, such tools usually do not address the complexity required by sophisticated games [1]. On the other hand, game engine tools are generally more powerful than authoring tools and they are the mainstream tools to create commercial games. But they are usually huge, complex, and sometimes lack usability, especially for beginners or amateurs. This makes the learning curve steep and using the tool cost expensive. What is more, both authoring tools and engine tools target pre-defined domains. Some emerging or innovative domains like pervasive games and education games may not be able to benefit from them. To ease the game creation especially for such specific domains, we propose an approach named Game Creation with Customized Tools (GCCT). By using GCCT, developers create tools according to specific domain requirements first, and then create games with these tools. GCCT is derived from the general Model Driven Development (MDD) approach [2] which has been proven effective in many domains [3, 4] to achieve higher productivity, shortened development cycle, and better software quality. The success of MDD was primarily achieved by providing high level and domain-specific abstractions (as the

K. Chorianopoulos et al. (Eds.): ICEC 2015, LNCS 9353, pp. 451–456, 2015.
DOI: 10.1007/978-3-319-24589-8_39

base of easy user interfaces) as well as code automation. Despite the apparent fitness and strength, involving MDD may also bring risks. For instance, MDD is not easy and it imposes high development cost for (domain specific) tools development in addition to the game development [5]. Previous research about applying MDD for game development did not address this issue in depth. To alleviate this, some enhancements were made in GCCT, and two case studies were performed to evaluate the cost and the productivity when involving GCCT.

The remainder of this paper is organized as below. We illustrate the GCCT approach in Section 2. Section 3 introduces the settings and results of case studies. In Section 4 we talk about practical lessons we have learned from the case studies. Finally in Section 5, we introduce possible future works and conclude the paper.

2 GCCT Formalism

As mentioned earlier, the motivation behind GCCT is that we create tools according to the specific domain requirements first, and then create games with these tools. Compared with pre-defined and mostly fixed authoring tools or engine tools, tools in GCCT are highly customizable. The overall process of GCCT is shown in Fig. 1. As presented in the figure, the tools customization in GCCT consists of three parts: game feature customization, game editor customization, and game code generator customization. While game feature customization defines which game features should be supported in the tools (editor, code generator, etc.) and needs to be done according to the specific project/domain requirements firstly, game editor customization and game code customization can be done afterwards based on the game feature set that has been decided. For game editors, we select the style (textual, tree-based, table-based, diagrammatical, or graphical). And based on the style, we connect editor elements to the game feature elements. For game code generators, we define the rules regarding how code snippets should be generated according to the game features. The tools customization is formalized according to traditions of MDD approaches [6].

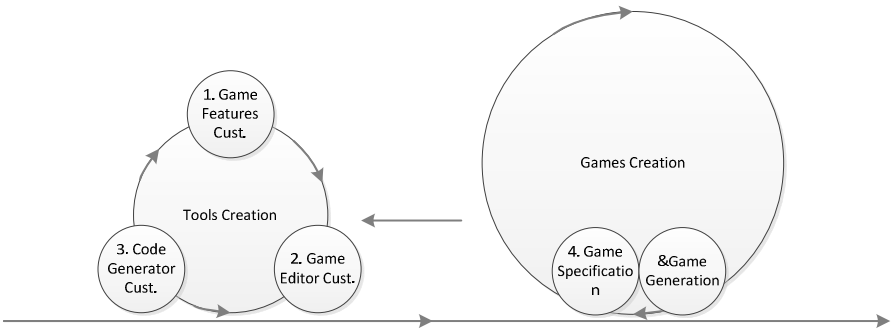

Fig. 1. GCCT Process View

In addition to the tools customization/creation, another part of GCCT is games creation. As shown in Fig.1, both parts are highly iterative. The necessity of intensive iterations in GCCT comes from the requirements of both the game domain and the MDD domain [2, 7, 8]. Although both parts are highly iterative, different parts are focused on when the projects progress. In the earlier stage of a project, usually more iterations of tools customization are performed, while several iterations of game creation may be used in order to validate the tools and provide feedback to improve the tools. In the latter stage of the projects, fewer iterations of tools creation may be performed because the tools have grown to be quite mature. And more games are developed in an iterative way. In order to fully play the strengths of MDD, more specialization and adaptation are done in GCCT in order to lower the technical barrier and the upfront and general cost. The main improvements are listed below, and the details were reported in [9].

- Instead of introducing a new Domain Analysis (DA) task [5], we propose to structure existing game design and level design tasks/documents in traditional game development to produce the DA outputs.
- Regulate and accelerate the DA task based on predefined domain vocabularies/ontologies (PerGO is proposed to be used for pervasive games domain [10]).
- Reuse existing working prototypes to construct the template of code generator.
- Utilize the state of the art and highly integrated language workbench tools [11].

3 Case Studies

Instead of addressing conventional computer games, we focused on the pervasive game domain in our research. This is mainly because pervasive games are innovative and there are no suitable authoring tools available for creating such games. Pervasive games have emerged during the last ten years. Such games involve more physical and social elements into the game, and blend game and everyday life by providing game experience all the time and everywhere [12]. The goals of performing the case studies include:

- To gain practical experiences of performing GCCT in general,
- To collect real project data regarding to the cost (for both tools and game software), and
- To gain lessons to reduce the cost.

We performed two case studies for this research. Prior to them, several pilot projects were carried out to get us familiar with model driven techniques as well as tools and libraries that would be used. We also gained some initial experience about how to perform MDD approaches in a more compact and efficient manner. Based on our experience, we consolidated and formalized the GCCT approach. Then we performed another two case studies with Realcoins and Realpacman to evaluate the GCCT approach. While Realpacman can be thought of a location-based variation of the traditional Pac-man game, Realcoins is a location-based treasure-hunting game

which has been reported in detail in [9]. These two case studies were using similar settings. For Realcoins, we only recorded basic cost data, while for the latter Realpacman, we recorded detailed cost distribution for major tasks in addition to the basic cost data. This is because after performing the Realcoins case, we found that the actual productivity was decided by many factors, and identifying such critical factors was important in order to control cost and improve productivity.

Lines of Code (LoC) and Hours (Hrs) are used for evaluating cost and productivity [1, 13, 14]. Table 2 and Table 3 present the LoC and Hrs used for the two case studies. For models, we also recorded LoC to quantitatively estimate the workload. LoC of a model was got by counting the lines of code in the corresponding textual specification of the model (.ecore files in our case). The productivity increase is calculated based on the productivity of creating games in a manual way. We call the approach of manually creating games as Game Creation from Scratch (GCS). In Table 2 and Table 3, the gray cells indicate one-time costs of GCCT, while the dotted cells indicate repetitive costs of GCCT (for each game instance).

Table 1. Hours used by Realcoins and Realpacman

		RealCoins		RealPacman	
		Hrs	**Percentage of Manual**	**Hrs**	**Percentage of Manual**
GCS (Manual)		14	100%	8.267	100%
GCCT	**Tools**	11	78.6%	5.3	64.1%
	Game	0.75	5.4%	0.55	6.7%

Table 2. LoC used by Realcoins and Realpacman

		RealCoins		RealPacman	
		LoC	**Percentage of Manual**	**LoC**	**Percentage of Manual**
GCS (Manual)		1263	100%	363	100%
GCCT	**Tools**	255	20.1%	167	46.0%
	Game	59	4.67%	93	25.6%

From Table 2 and Table 3, we can see that by using GCCT, 5.4% and 6.7% of the manual development time was used for each new game instance. And the corresponding LoCs needed for GCCT were 4.67% and 25.6% of the manual ones. In another word, to develop every new game instance, using GCCT can be 4-20 times faster/cheaper than using GCS. The cost saving was very obvious. Although there was an upfront cost for the GCCT approach, it was relatively small. From the data in the two tables, this extra cost to develop GCCT tools required around 60%-80% time or 20%-50% LoC of manually developing one game instance. That is to say, this one-time cost might be paid back after two or three game instances were developed.

Despite the promising results in general, we also noticed that the data varied severely from case to case as a result of different domain complexity and practical factors. In the next section, we further introduce some practical lessons we have

learned from the case studies in order to perform such a MDD approach in an efficient way.

4 Lessons Learnt

MDD is not a silver bullet [5]. The decision must be carefully made about whether or not to adopt MDD according to various aspects of the domains/projects. The actual productivity increase brought by involving MDD approaches may be influenced by many factors. From the case studies, we got some practical lessons about saving cost and therefore achieving a higher productivity increase.

1. *Start with stable but extensible meta-model*. This is important to make sure the tools are built up in an incremental way without frequent re-construction. During the pilot case studies, we did not have a common basic structure for the meta-models, which sometimes made it difficult to implement new features during the incremental process. And we had to re-create everything which made the overall development cycle much longer accordingly. But when we applied PerGO (in Realcoins and Realpacman) to build an extensible meta-model structure, such situation did not appear frequently.
2. *Make abstractions quickly with knowledge engineering*. Using PerGO in case studies helped us streamline domain analysis and define the meta-model quickly.
3. *Use easy-to-implement abstractions.* If it is difficult to find corresponding constructs to implement the meta-model concepts, building domain specific libraries will become much more difficult and time consuming. In addition, specifying the relationship among them in the code template gets even more difficult.
4. *Reuse codes of working prototypes to construct generators.* A working starting point made creating generator quicker and of less bugs.
5. *Generate complete codes*. If some codes (even several lines of code) need to be manually written after other parts have been automatically generated, it means this part of work will be repeated for all the iterations. Generating complete codes reduces the routine cost of iterations and makes the overall process much agile.
6. *Regulate structured process within one iteration*. The process should include both game design tasks and MDD related tasks. A reasonable and regulated process helps to reuse resources in an efficient way, smooth the task transition, and lower the routine cost for every iteration.
7. *Adopt highly automatic language workbench tools.* By doing so, many routine tasks like generating concept classes can be automated.
8. *Adopt highly integrated development environment*. Highly integrated development environment helps avoid some unnecessary tasks (like copying source codes from one tool to another, or changing formats of data due to different requirements from tools), and makes the overall process smoother.

5 Future Work and Conclusion

In this paper we have introduced GCCT, an enhanced MDD approach for computer game creation. We evaluated on the cost and the productivity when involving GCCT through case studies. In order to perform efficient MDD approaches in other projects, we presented some practical lessons we have gained from the cases. Consequently, we have identified some future work to address the cost issues: 1) we will explore major impact factors for the actual productivity increase of involving MDD approaches like GCCT; 2) we will observe how the cost of tools development and games development evolve as the project proceeds within iterations.

References

1. Furtado, A.W., Santos, A.L.: Using domain-specific modeling towards computer games development industrialization. In: 6th OOPSLA Workshop on Domain-Specific Modeling (DSM 2006). Citeseer (2006)
2. Kelly, S., Tolvanen, J.-P.: Domain-Specific Modeling Enabling Full Code Generation. John Wiley & Sons, Inc. (2008)
3. Hutchinson, J., Whittle, J., Rouncefield, M.: Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. Science of Computer Programming 89, 144–161 (2014)
4. Hussmann, H., Meixner, G., Zuehlke, D.: Model-Driven Development of Advanced User Interfaces. Springer Science & Business Media (2011)
5. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. 37(4), 316–344 (2005)
6. Guo, H., Gao, S., Krogstie, J., Trætteberg, H.: An Evaluation of an Enhanced Model Driven Approach for Computer Game Creation. In: Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q. (eds.) BPMDS 2015 and EMMSAD 2015. LNBIP, vol. 214, pp. 499–508. Springer, Heidelberg (2015)
7. Adams, E.: Fundamentals of game design. New Riders (2010)
8. Gal, V., et al.: Writing for video games. In: Proceedings of the Laval Virtual, IVRC (2002)
9. Guo, H., et al.: RealCoins: A Case Study of Enhanced Model Driven Development for Pervasive Game. International Journal of Multimedia and Ubiquitous Engineering 10(5), 395–410 (2015)
10. Guo, H., Trætteberg, H., Wang, A.I., Gao, S.: PerGO: An Ontology towards Model Driven Pervasive Game Development. In: Meersman, R., et al. (eds.) Workshops at the OTM 2014. LNCS, vol. 8842, pp. 651–654. Springer, Heidelberg (2014)
11. Fowler, M.: Language workbenches: The killer-app for domain specific languages (2005)
12. Guo, H., et al.: TeMPS: A Conceptual Framework for Pervasive and Social Games. In: 2010 IEEE 3rd International Conference on Digital Game and Intelligent Toy Enhanced Learning, pp. 31–37. IEEE Press (2010)
13. Hernandez, F.E., Ortega, F.R.: Eberos GML2D: a graphical domain-specific language for modeling 2D video games. In: Proceedings of the 10th Workshop on Domain-Specific Modeling, Reno, Nevada, p. 1. ACM (2010)
14. Reyno, E.M., Carsí Cubel, J.Á.: Automatic prototyping in model-driven game development. Computers in Entertainment (CIE) 7(2), 29 (2009)