

# Specifying a Software Factory for 2D Adventure Computer Games Development

André W. B. Furtado, André L. M. Santos  
Microsoft Innovation Center at Recife,  
Rua do Apolo, 181,  
Bairro do Recife,  
CEP 50030-220, Recife/PE/Brazil  
+55 (81) 34198134  
{awbf,alms}@cin.ufpe.br

Rogério Panigassi, Jorge L. R. Becerra  
Escola Politécnica da Universidade de São Paulo  
Av. Prof. Almeida Prado, Travessa 2,  
Cidade Universitária  
CEP 05508-900, São Paulo/SP/Brazil  
+55 (11) 3091-5200  
{rogerio.panigassi,jorge.becerra}@poli.usp.br

## 1. INTRODUCTION

Digital games are one of the most profitable industries in the world, being a match even for the movie and music industries. However, software development industrialization, an upcoming tendency entailed by the exponential growth of the total global demand for software, will present many new challenges to game development. Studies reveal that there is evidence that the current development paradigm is near its end, and that a new paradigm is needed to support the next leap forward in software development technology [1].

For example, although game engines brought the benefits of Software Engineering and object-oriented technologies towards game development automation, the abstraction level provided by them could be made less complex to consume by means of language-based tools, the use of visual models as first-class citizens (in the same way as source code) and a better integration with development processes.

This research, therefore, explores the integration between game development, an inherently creative discipline, with software factories, which are concerned with turning the current software development paradigm, based on craftsmanship, into a manufacturing process. A specification (schema) for a simple software factory named *SharpLudus* is described, illustrating how industrialization can be applied to digital games development. Although a specific domain (based on the 2D adventure game genre) was chosen for the *SharpLudus* product line, the methodology presented in the following sections can also be applied to the creation of software factories targeted at other game domains as well.

The proposal includes both product line analysis and product line design tasks. It defines an architecture for the target software product family, the product development process and a plan for using tools to automate parts of the product development process. The final intention is to empower game developers and designers to work more productively, with a higher level of abstraction and closer to their application domain.

## 2. PRODUCT LINE ANALYSIS

Product line analysis is the first group of tasks to be performed in the construction of a software factory. The purpose of this activity is to decide *what* products the software factory will produce. The focus of product line analysis is describing the commonalities and variabilities that characterize the product family.

The great diversity of games created so far has turned the digital games universe into a very broad domain. Therefore, creating a software factory targeted at computer games development in general, ranging from 2D platform games to 3D flight simulators, constitutes a too broad and ineffective endeavor. In such a scenario, the production process and its tools would not be able to fully exploit factory benefits such as component reuse and assemblage. In other words, a narrower subset of games should be chosen.

One of the most often used attempts to classify computer games is to define game *genres*, which together compose a game *taxonomy* [2]. Some of the most popular game genres [2, 3, 4] are summarized in Table 1.

**Table 1 – Popular game genres**

| Genre Name   | Description  | Examples  |
|--------------|--|---|
| Adventure    | Games which are set in a “world” usually made up of multiple, connected rooms or screens, involving an objective which is more complex than simply catching, shooting, capturing, or escaping, although completion of the objective may involve several or all of these.                 | Berzerk, Adventure (Atari), Myst, Tomb Raider, Space Quest, Indiana Jones |
| Board        | Adaptation of existing board games or games which are similar to board games in their design and play even if they did not previously exist as board games.  | Backgammon, Othello, Checkers, Chess                                      |
| Fighting     | Games involving characters who fight usually hand-to-hand, in one-to-one combat situations. In most of these games, the fighters are represented as humans or anthropomorphic characters.  | Street Fighter, Mortal Kombat, Dragon Ball Z                              |
| Platform     | Consists of animated objects running, climbing and jumping on platforms. Characters and settings are seen in side view as opposed to top view, and most games scroll the screen while the main character moves.  | Super Mario Bros., Alex Kid, Comander Keen                                |
| Role-Playing | Games in which players create or take on a character represented by various statistics, which may even include a developed persona.  | Final Fantasy, Dungeons & Dragons, Diablo                                 |
| Shooter      | Games involving shooting at, and often destroying, a series of opponents or objects, usually requiring quick reflexes.   | Doom, Half-Life, Halo   |
| Simulation   | Games or programs which attempt to simulate a realistic situation, for the purpose of training, and usually the development of some physical skill such as steering (as in driving and flight simulators).   | Flight Simulator, Apache, F-22, The Sims                                  |
| Sports       | Games which are adaptations of existing sports or variations of them.  | NHL Hockey, FIFA Soccer, NBA Basketball, Top Spin                         |
| Strategy     | Games which require planning, complex decisions and balancing the use of limited resources towards a higher-level goal (improve a city, evolve a civilization, etc.), while dealing with internal forces (crime, pollution, etc.) or external forces (natural disasters, enemies, etc.). | Age of Empires, WarCraft, Civilization, SimCity                           |

Defining genres, however, can be a quite difficult task as many people have different opinions on the meaning of a genre or various ways of stereotyping them [5]. Likewise, it is not rare for a game to fall into more than one category. Some authors even argue that describing different types of games requires different dimensions of distinctions (narratology, ludology, simulation, gambling, etc.), i. e., orthogonal taxonomies which allow design concerns to be separated [6].

This research, therefore, suggests that in spite of solely relying on a game genre name to define a software factory product line, a description of what is understood from such genre should also be provided, as well as any relevant information such as dimension (2D, 2D ½, 3D), sound support, input handling, networking support and so on. A suggestion of a product line definition template, filled with SharpLudus information, is presented in Table 2.

**Table 2 – SharpLudus Product Line Definition**

| <b>SharpLudus Game Software Factory - Product Line Definition</b>   |  |
|---|--|
| <b>Related game genre(s):</b> adventure   |  |
| <b>Description:</b> The factory will produce computer games in which the player control a main character in a world composed by connected rooms. Rooms may contain items to be collected, such as keys and weapons. Enemies may also be present in a room; they must be avoided or defeated. Victory condition is specified by the game designer (a specific room is reached, a number of enemies is defeated, an object is collected, etc.). |  |
| <b>Target Platforms:</b> PCs and mobiles devices (PocketPCs and smart phones)   |  |
| <b>Feature Overview</b>   |  |
| <b>Feature</b>  | <b>Description</b>   |
| Dimensionality  | Two-dimensional (2D). World rooms are viewed from above.   |
| User interface  | Information display screens containing texts, radio buttons and graphical elements are supported. HUDs (heads-up display) can also be configured and displayed.  |
| Game flow   | Each game should have, at least, a main character, an introduction screen, one room and a game over screen (this last one is reached when the number of lives of the main character becomes zero).     |
| Sound/Music   | Games will be able to reproduce sound effects (wav files) as event reactions. Background music (mp3 files) can be associated with game rooms or information display screens.                           |
| Input handling  | Keyboard only.   |
| Networking  | High scores can be uploaded to and retrieved from a web server.  |
| Artificial Intelligence   | Enemies can be set to chase the player within a room. More elaborated behaviors can be created visually by combining predefined event triggers and event reactions, or programmatically by developers. |
| Multiplayer   | Online multiplayer is not supported by the factory. Event triggers and reactions can be combined, however, to allow two-player mode in a single computer.  |
| End-user editors  | Not supported by the factory. Once created, a game cannot be customized by its players.  |

As it can be observed, games that can be produced by the SharpLudus factory are somewhat simple. Although one may argue that such games do not present the characteristics of successful titles, this does not invalidate the research. Although two-dimensional games are more and more being replaced by 3D games, they are still appreciated by several people [7]. Even today, the game industry still invests in 2D games. Many publishers, for example, have recently released nostalgic compilations of many successful 2D titles, such as Pac Man, Street Fighter, Dig Dug and so on. Examples of such compilations are the Capcom Classics Collection, Tecmo Arcade Classics, Taito Legends and Namco Museum 50<sup>th</sup> Anniversary [8].

Moreover, there are always opportunities and markets for “low-end games”, such as games for cell-phones or old arcade games. Low-budget, value-priced games (those selling in the U\$6-U\$15 range) seem to have a longer shelf-life than the big budget games [9]. The per-game profit is lower, of course, but the development and distribution costs are also much lower. Such a reality is already explored by some game industry initiatives, such as the Xbox Live Arcade [10], in which developers are able to create and sell simple Xbox games (traditional card games, nostalgia arcade games, puzzles and trivia quizzes). On the other hand,

John Buchanan, Electronic Arts CTO, reveals in his keynote “Experiences, Stories and Video Games” [11] that current complex 3D games fail to provide immersion to game players, since the expectations around them are too high. He points out, for example, that users still prefer the first, two-dimensional version of NHL Hockey (a successful Electronic Arts title) in comparison with the latest 3D versions. In other words, the higher the graphics complexity introduced by a game, the harder its suspension of disbelief<sup>1</sup>.

Finally, it is worth noticing that real commercial computer games demand a development effort which is out of the scope of this research. More than two years are usually required for completing such kind of games, and about 30 people are involved, which can equate to an investment of about U\$9 million.

### 3. PRODUCT LINE DESIGN

The product line design macro-activity defines *how* a software factory will develop products within its scope. It defines an architecture for the target software product family, the product development process, and a plan for using tools to automate parts of the product development process.

#### 3.1 SharpLudus Software Factory Schema

Software factory assets (designers, domain-specific languages, frameworks, tools, etc.) must be properly specified and the software factory schema (specification) can finally be created. A factory schema contains a graph of viewpoints, each one defining part of the system at a specific level of abstraction and at a specific point in the software life-cycle. Factory assets are associated to viewpoints as well.

Figure 1 presents the SharpLudus software factory schema. It shows a graph whose nodes are viewpoints and whose edges are computable relationships between viewpoints (mappings). Related viewpoints are grouped in clusters to better indicate their purpose and positioning in the development workflow. Some process artifacts (such as the Test Plan, Master Schedule, etc.) were suppressed from the schema.

The factory schema reveals that the first factory asset to be used is an integrated development environment (IDE), such as the Microsoft Visual Studio .NET [12] or Eclipse [13], customized with a “new project wizard”, targeted at the creation of factory games. Such a wizard must be launched when the original game idea is conceived. It collects information to create the first version of three development artifacts: the Game Design, the Business Case and the Glossary.

The Game Design is the most important artifact of the requirements architecture cluster. It gathers information about the goals of the game under development, its motivation, target platform, story, characters overview, game environments, scoring formula, etc. The Business Case, on the other hand, is used to provide information regarding cost-benefit analysis, market viability, competitors (similar games) and so on. Finally, the Glossary describes terms and definitions for concepts related to the game under development.

By following the Game Design artifact, the team should use graphical tools and sound/music processors to create the art conception and the sound conception for the game under development. Such conceptions are then exported to the SharpLudus sprite designer and audio component designer, respectively, in order to allow the creation of entity sprites, graphical elements of info display screens, room tiles and the game HUD, as well as sound effects (assigned to game event reactions) and background music (assigned to game states, which can be rooms of information display screens). Other factory assets, such as the entity designer and event designer, receive the Game Design as input and allow developers to de-

---

<sup>1</sup> Suspension of disbelief is a willingness of a reader or viewer to suspend his or her critical faculties to the extent of ignoring inconsistencies and unreality so as to enjoy a work of fiction.

fine the game entities (main character, non-playable characters and items) and rules (events) that govern the game world. The use of the Visual Studio Tools for the Microsoft Office System technology (VSTO) [14] also makes it possible for the factory to programmatically extract information from the Game Design document and automatically fill part of the game specification.

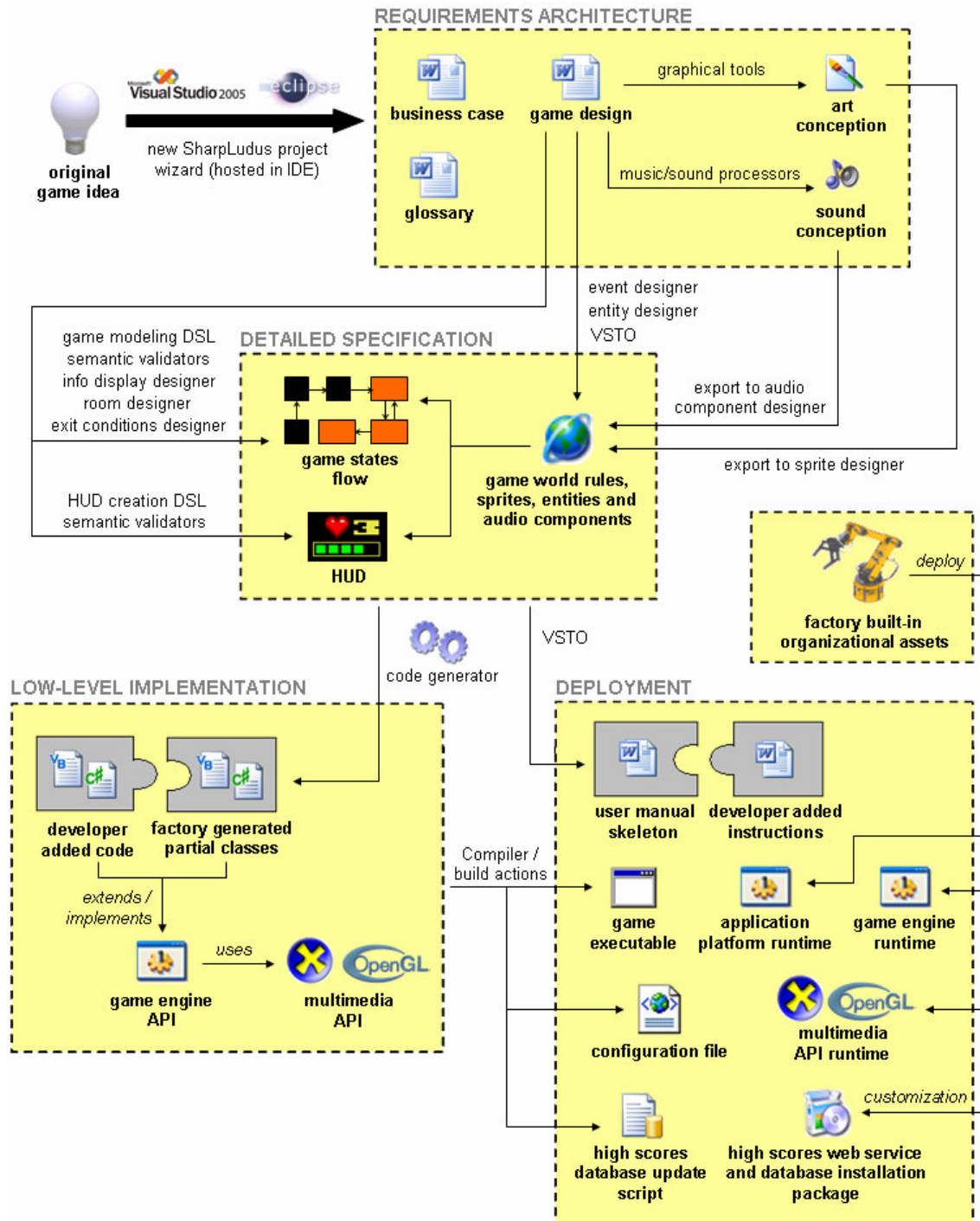


Figure 1 – SharpLudus software factory schema

The SharpLudus software factory also provides to developers two visual domain-specific languages (DSLs) [15] as assets. The first one is the Game Modeling DSL, which together with the room designer and the info display designer allows the specification of the game states flow (info display screens, rooms and their exit conditions). Since game states are also related to entities, background music and game configuration (such as its resolution) the game states flow also receive the overall game specification (sprites, entities, audio components, etc.) as input. The second domain-specific language is the HUD creation DSL, which allows developers to specify how useful game information (score, remaining lives, hit points, etc.) will be presented to the player by means of a heads-up display. Both DSLs are provided with validators to ensure that semantic errors are caught in design time and shown in the IDE error list.

The SharpLudus detailed game specification, contrary to common game development approaches, is a set of *live* artifacts. This means that they are not only used for documentation, but they can be transformed into other artifacts by means of automation assets. For example, the VSTO technology can be used to create a User Manual skeleton with information extracted from the game specification, while code generators associated to the DSLs can be used to automatically create the majority of the game implementation code. Developers, however, can add their own code to the solution since the factory generated code provides extensibility mechanisms such as partial classes<sup>2</sup> and classes which are just ready for customization (for example, special classes for providing custom event triggers and custom event reactions).

Both the factory generated code and the developer added code interacts with a game engine API, which is chosen in the new SharpLudus project wizard. The game engine API uses a Multimedia API, such as DirectX or OpenGL, which is also chosen in the project wizard. Once the solution implementation is compiled, the factory generates the game executable file and a XML configuration file through which the high scores web server address and custom developer configuration can be specified. A database update script (such as a .sql file) is also generated, in order to update the high scores server with the new game information. Finally, built-in factory organizational assets, such as the runtimes of the game engine and the multimedia API chosen, are automatically made available by the factory.

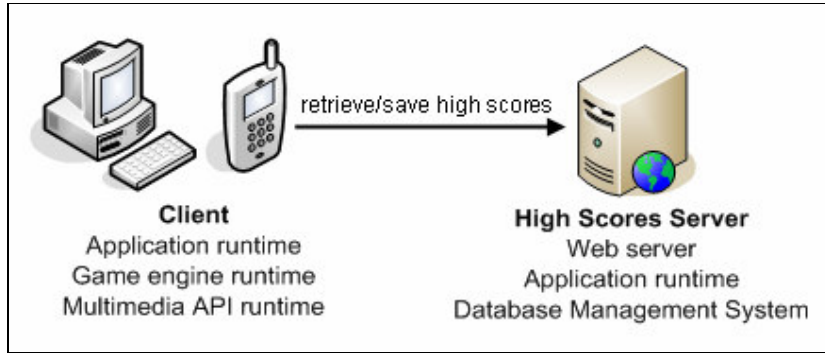
### 3.2 Product Line Architecture

The software factory product line architecture is one of the most important assets produced by product line development. It describes the common high-level design features of the products that will be produced by the product line and distinguish between common and variable design features.

For the SharpLudus software factory, a top-level network topology is presented in Figure 2. It is actually very simple, comprising only two components: the target client where the created game is supposed to be played and a web server that hosts a web service [16] which is responsible for storing and retrieving scores of games created through the factory. Regarding architecture extensibility, the required software for each component is presented generically. When the software factory template is instantiated, such generic software is replaced by concrete specializations. For example, client software could include the Microsoft .NET Framework 2.0 runtime, the assemblies of the DigiPen game engine [17] and the DirectX 9 runtime [18], while server software could include Microsoft Internet Information Services [19], SQL Server 2005 [20] and the .NET Framework 2.0 runtime as well.

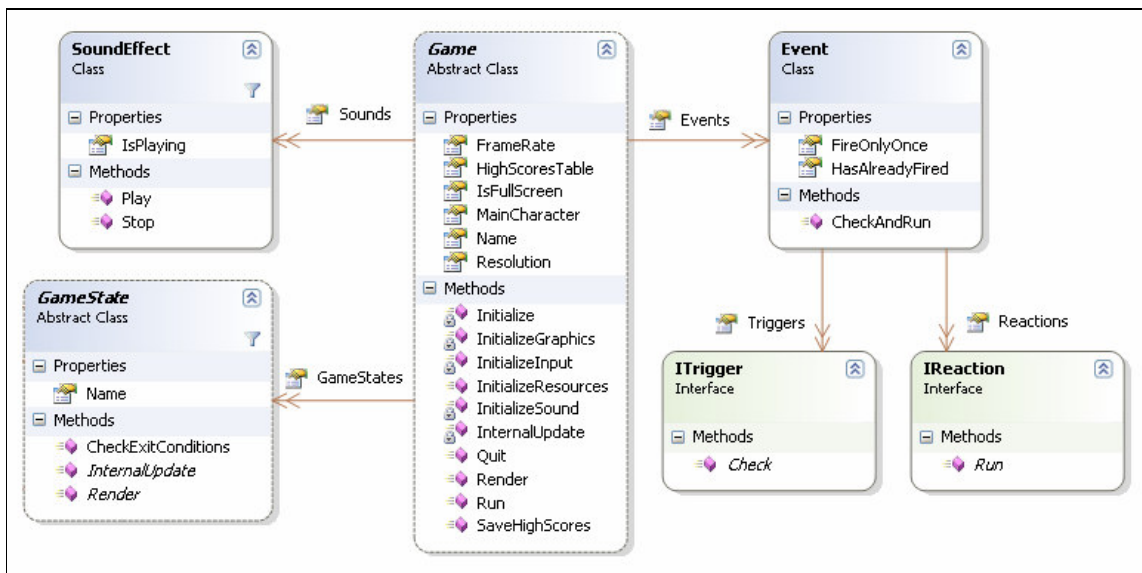
---

<sup>2</sup> The concept of partial classes makes it possible to split the implementation of a single class in two files.



**Figure 2 – SharpLudus top-level network topology**

The low-level architecture of games produced by the SharpLudus factory is presented in the following three figures, not in only one, due to space constraints. Figure 3 presents the main *Game* class and how game events and sound effects are implemented. Figure 4 details the implementation of game states, their exit conditions and background music. Finally, Figure 5 provides an implementation overview for entities and sprites. As it may be noticed, such architecture reveals that game implementation paradigm is not an extensibility point of the SharpLudus software factory, since it is focused in creating games whose implementation is object-oriented.



**Figure 3 – SharpLudus game architecture (1): the Game class and its relations**

The main extensibility mechanisms of the proposed architecture are class inheritance and interface implementation. Factory games should inherit from the *Game* class and specify their own configuration. The factory provides some built-in implementations of the *ITrigger* and *IReaction* interfaces (such as *HasItemInInventoryTrigger* and *SetGameStateReaction*) but the developer is free to create its own triggers and reactions. Finally, many of the classes, such as *Entity*, *Sprite* and *Room*, are actually partial classes. Therefore, users can extend their behavior by adding their own code to the solution implementation.

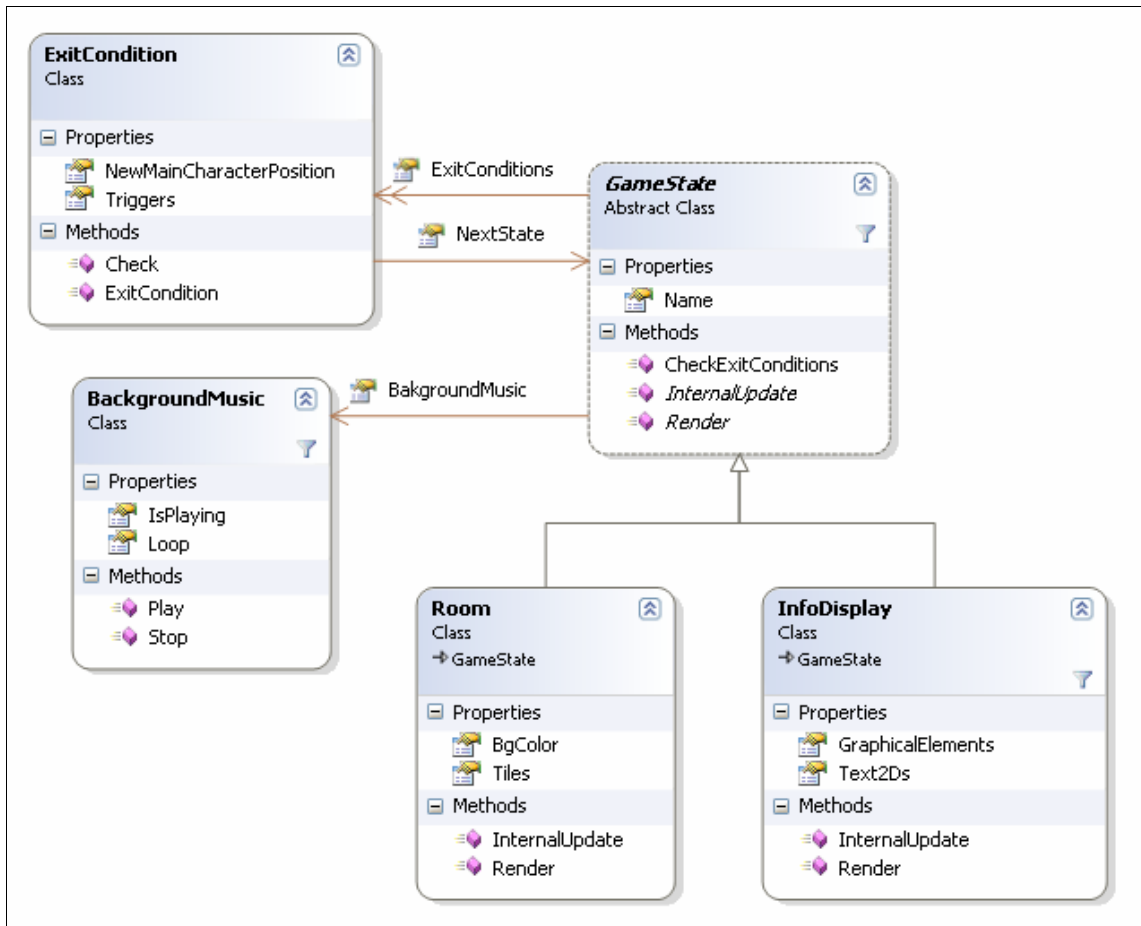


Figure 4 – SharpLudus game architecture (2): the *GameState* class and its relations

### 3.3 Product Development Process Definition

Process definition describes the common and variable features of the product development process. It produces a process that identifies the production assets and explains when, how, and by whom they are used to develop the product family members. It may describe the activities performed, the artifacts they produce or consume, and guidelines, conventions, and policies for using the production assets.

The complete definition of a software development process is not the focus of this research. However, an overview of how such process would look like, by means of a Team Model and a Process Model specification, is provided. The process is based on the Microsoft Solutions Framework for Agile Software Development [21].

Regarding the Team Model (i.e., the assignment of responsibilities to the individuals involved in the software lifecycle) the SharpLudus software factory suggests 9 roles, as described in Table 3. Role responsibilities and related factory assets / artifacts are provided as well.

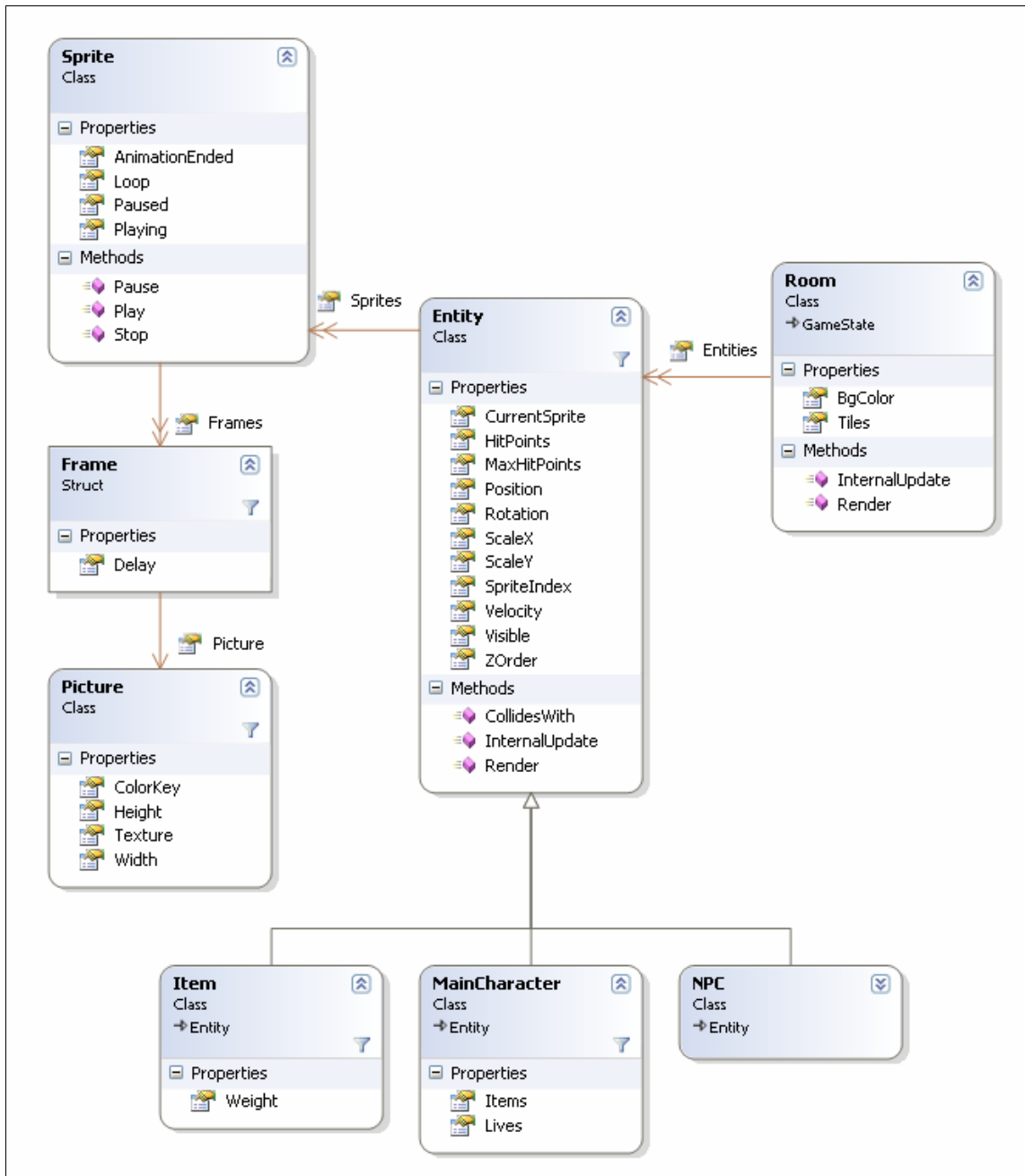


Figure 5 – SharpLudus game architecture (3): the Entity class and its relations

**Table 3 – SharpLudus software factory Team Model**

| <b>Role</b>        | <b>Responsibility</b>  | <b>Related Assets / Artifacts</b>  |
|--------------------|--|--|
| Game Design        | Elaborate the game story and goals; specify game configuration, states, entities and rules   | New project wizard, Game Design, Glossary, event designer, entity designer, Game Modeling DSL, HUD creation DSL, semantic validators, info display designer, room designer, exit conditions designer |
| Product Management | Understand, communicate and ensure success from the standpoint of publishers requesting the game.  | Business case  |
| Program Management | Meet the goal of delivering the solution within project constraints (scope, resources and schedule).   | Process management artifacts (Master Project Plan, Project Schedule, etc.)   |
| Art and Animation  | Create, edit and/or request acquisition of game graphical resources, such as sprites and Heads-up Display  | Art conception, sprite designer  |
| Audio              | Compose, edit, and/or request acquisition of game audio resources, such as sound effects and background music  | Sound conception, audio component designer   |
| Development        | Deal with services, technical and standards with which the solution will interoperate; build more complex game logic; is the primary consultant on technical decisions | Product line architecture, code generator, developer added code, game engines, multimedia APIs, unit tests   |
| Test               | Anticipate, look for, and report on any issues that diminish the solution quality in the eyes of the users or customers.   | Test plans, test scripts, test reports, bug tracking   |
| Release Operations | Ensure timely readiness and compatibility of the infrastructure for the solution   | Database scripts, installation packages, client / server infra-structure setup   |
| User Experience    | Understand players' context as a whole, appreciate any subtleties of their needs, and ensure that the whole team is conscious of usability from their eyes.            | User Manual, acceptance tests, player feedback   |

Regarding the Process Model (i.e., the structure and flow of activities that must be carried out during product development) the SharpLudus software factory suggests that micro-processes [22] are assigned to each viewpoint, describing the development of conforming views, and by defining constraints like pre-conditions that must be satisfied before a view is produced, post-conditions that must be satisfied after it is produced, and invariants that must hold when the views have stabilized. Such micro-processes should be supported by the host IDE by means of context-sensitive help and automation technologies such as Guidance Automation Toolkit (GATs) [23], which make reusable code and pattern assets directly available in an IDE. Such an approach helps the development process to be less prescriptive and more practical, focused on the team needs for the task currently in execution. Examples of this approach follow below:

- When adding a custom event trigger to a game, developers can press the F1 key to make the IDE show examples of custom triggers, code templates and worries that they must be aware of.
- If developers want to, they can order the insertion of a custom trigger code template into the code under development. Wherever the event trigger is needed, a GAT recipe will automatically launch actions to ensure that the Singleton design pattern [24] is implemented, in order to avoid the trigger to be unnecessarily instantiated multiple times.
- When the F1 key is pressed while the game designer is specifying the game states flow, however, different guidance will be presented by the IDE: the documentation of the SharpLudus Game Modeling DSL, as well as examples and ready-to-use templates.

Finally, the SharpLudus software factory Process Model also suggests that iterations are carried out periodically, according to the Table 4.

**Table 4 – Product line development iterations**

| Iteration               | Purpose  |
|-------------------------|--|
| 0                       | Develop requirements architecture foundation (Business Case, Game Design)  |
| 1                       | Create project plans; initial art/sound conceptions and game specification (rules, entities, states flow, HUD, etc.)     |
| 2..n (repeat as needed) | Implement custom code; refine game specification and art/sound conceptions; test; retrieve and analyze players' feedback |
| n + 1                   | Package game for deployment; deploy  |

### 3.4 Product Line Requirements Mapping

The goal of product line requirements mapping is to map variabilities in the product line requirements onto variability mechanisms in the production assets, especially the product line architecture, the implementation components, and the product development process. The resulting mapping for the SharpLudus software factory is presented in Table 5.

## 4. CONCLUSIONS

This paper presented a reusable methodology for creating software factories targeted at computer games, encompassing a concrete approach for executing product line definition and product line design activities. The definition of assets targeted at a specific game domain and a complete software factory schema were de-scribed as well.

Since the proposed approach and tools are focused on a specific domain, they may not be suitable to other types of game development contexts. Therefore, one interesting future work is the creation, based on previously acquired knowledge, of other factories targeted at other game genres, such as racing games or first-person shooters. Some domain concepts, factory designers, semantic validation rules and excerpts of the code generator may be reused, while others will need to be recreated.

Table 5 – Feature/asset map for the SharpLudus production line

| Feature / Assets                            | Development process | New project wizard | Event designer | Entity / Sprite designers | Audio component designer | VSTO | Game Modeling DSL | HUD creation DSL | Semantic validators | Info display / Room designers | Product line architecture / user added code | Code generator | Compiler / build actions | High Scores server customization |
|---|---------------------|--------------------|----------------|---------------------------|--------------------------|------|-------------------|------------------|---------------------|-------------------------------|---|----------------|--------------------------|----------------------------------|
| Set generated code language                 | X                   | X                  |                |                           |                          |      |                   |                  |                     |                               | X   | X              | X                        |                                  |
| Use a specific game engine                  | X                   | X                  |                |                           |                          |      |                   |                  |                     |                               | X   | X              | X                        |                                  |
| Use a specific multimedia API               | X                   | X                  |                |                           |                          |      |                   |                  |                     |                               | X   | X              | X                        |                                  |
| Define target platform (PC / Mobile / etc.) | X                   | X                  | X              |                           |                          | X    | X                 | X                | X                   | X                             | X   | X              | X                        |                                  |
| Specify custom event triggers and reactions | X                   |                    | X              |                           |                          |      |                   |                  |                     |                               | X   |                | X                        |                                  |
| Design game sprites, entities, and audio    | X                   |                    |                | X                         | X                        |      |                   |                  |                     |                               |   |                |                          |                                  |
| Design Heads-up Display                     | X                   |                    |                |                           |                          |      |                   | X                | X                   |                               |   |                |                          |                                  |
| Specify game flow                           | X                   |                    |                |                           |                          |      | X                 |                  | X                   |                               |   |                |                          |                                  |
| Configure user manual                       | X                   | X                  |                |                           |                          | X    |                   |                  |                     |                               |   |                |                          |                                  |
| Specify server infra-structure              | X                   | X                  |                |                           |                          |      |                   |                  |                     |                               |   |                |                          | X                                |

New improvements added to SharpLudus are intended to be reported in the project website ([www.cin.ufpe.br/~sharpludus](http://www.cin.ufpe.br/~sharpludus)), which also provides factory resources and sample games for download. While the results obtained so far empirically shows that the SharpLudus factory is indeed an interesting approach, it is important to notice that deploying a complete software factory is also associated with some costs. Return of investment may arise only after a certain amount of games are produced. Besides that, despite being easy to use, software factories are complex to develop. They will certainly require a mindset evolution of the game development industry.

A final remark is that the presented proposal alone will not ensure the success of game development. In fact, no technology is a substitute for creativity and a good game design. Game industrialization, languages, frameworks and tools are means, not goals, targeted at the final purpose of making people have entertainment, fun and enjoy themselves. Players, not the game or its constituent technologies, should be the final focus of every new game development endeavor.

## 5. REFERENCES

- [1] Greenfield, J. et al., *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley & Sons, 2004;
- [2] Crawford, C. *A Taxonomy of Computer Games*, <http://www.vancouver.wsu.edu/fac/peabody/game-book/Chapter3.html>;
- [3] Sawyer, B. *The Getting Started Guide to Game Development FAQ*, <http://www.gamedev.net/reference/articles/article261.asp>;
- [4] Wolf, M. *Genre and the Video Game*, in Wolf M.J.P., *The Medium of the Video Game*, University of Texas Press, 2002;
- [5] Oxland, K. *Gamplay and Design*, London: Pearson Education, 2004;
- [6] Lindley, A. *Game Taxonomies: A High Level Framework for Game Analysis and Design*, [http://www.gamasutra.com/features/20031003/lindley\\_01.shtml](http://www.gamasutra.com/features/20031003/lindley_01.shtml);
- [7] Wiering, M. *The Clean Game Library*, MSc dissertation, University of Nijmegen, 1999;
- [8] Folha de São Paulo, *Collection packs bring arcade games back to videogames* (in Portuguese), Informatics, October 12, 2005;
- [9] Gruber, D. *The Business of Games*, <http://www.fastgraph.com/makegames/chapt4.html>;
- [10] Xbox.com, *Xbox Live Arcade*, <http://www.xbox.com/en-US/livearcade/default.htm>;
- [11] Buchanan, J. *Experiences, Stories and Video Games*, in Proceedings of WJogos 2005, SBGames 2005;
- [12] MSDN.com, *Visual Studio 2005 Team System: Overview*, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvsent/html/vsts-over.asp>;
- [13] Eclipse.org, *Eclipse Home*, <http://www.eclipse.org/>;
- [14] MSDN.com, *Visual Studio Tools for Office Developer Portal*, <http://msdn.microsoft.com/office/understanding/vsto/default.aspx>;
- [15] Deursen, A.; Klint, P.; Visser, J. *Domain-Specific Languages: An Annotated Bibliography*, <http://homepages.cwi.nl/~arie/papers/dslbib/>;
- [16] W3C, *Web Services*, <http://www.w3.org/2002/ws/>;
- [17] DigiPen.edu, *MSDN Webcast Archive - Video Game Development: Learn to Write C# the Fun Way*, <http://www.digipen.edu/webcast>;
- [18] *Microsoft DirectX*, <http://www.microsoft.com/directx>;
- [19] Microsoft.com, *Internet Information Services*, <http://www.microsoft.com/WindowsServer2003/iis/default.aspx>;
- [20] Microsoft.com, *Microsoft SQL Server*, <http://www.microsoft.com/sql/default.aspx>;
- [21] Microsoft.com, *MSF for Agile Software Development*, <http://msdn.microsoft.com/vstudio/teamsystem/msf/msfagile/>;
- [22] Institut für Informatik, Freie Universität Berlin, *Micro-process of software development*, <http://projects.mi.fu-berlin.de/w/bin/view/SE/MicroprocessHome>;
- [23] MSDN.com, *Introduction to the Guidance Automation Toolkit*, <http://lab.msdn.microsoft.com/teamsystem/workshop/gat/intro.aspx>;
- [24] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Longman, 1998;