Serious Games Architectures and Engines

Heinrich Söbke[1], Alexander Streicher[2]

[1] Bauhaus-Institute for Infrastructure Solutions (b.is), Bauhaus-Universität Weimar, Germany
heinrich.soebke@uni-weimar.de

[2] Fraunhofer IOSB, Karlsruhe, Germany
alexander.streicher@iosb.fraunhofer.de

Abstract. The term *Serious Game* includes a wide, heterogeneous field of digital games with varying purposes and objectives and for a multitude of different application areas. All in common is the underlying software. This chapter gives an overview on the technical aspects of serious games including their software architectures and engines. As the general topic is manifold and the technical aspects of serious game software are quite comprehensive, this chapter covers the basic principles of and requirements for serious game software. It depicts selected software architectures and provides examples for game engines including a description of selected components.

Keywords: Serious Games Architecture, Game Engine, Serious Game Development, Distributed Architecture, Game Component, Schema.

1 Introduction

What are serious games and how can they be categorized? Schmidt et al. [1] suggest a categorization according to the purpose of the game: they follow the work of Connolly et al. [2] and classify the purposes of a game as *Attention*, *Motivation*, *Knowledge or skill acquisition*, *Process Support*, *Joy/Playfulness* and *Information*. Michael & Chen [3] identify eight categories as markets for serious games: *Military*, *government*, *education*, *corporate games*, *healthcare*, *politics*, *religion* and *art*. These are only two of the suggested categorizations – Djaouti et al. [4] present a literature review of serious games categorization in their work to develop their G/P/S model. This model divides a serious game into **G**ame aspects and serious aspects (**P**urpose and **S**cope). Another classification has been defined by Ratan & Ritterfeld [5]. Their proposed dimensions to classify serious games comprise *Primary Educational Content*, *Primary Learning Principles*, *Target Age Group* and *Game Platform*.

The amount and depth of all those classifications can be taken as an indicator for the great diversity in the field of serious games. Sophisticated distributed virtual training systems for military operations [6] are subsumed under this term as it is done for multiplayer online games [7], or for a simple gamified quiz app [8]. Software is a common component of all these games. This requires a non-trivial software development pro-

cess. Such a process, the chosen software architecture and the employed software development tools depend on the planned serious game. Obviously, neither a universal serious game template nor a universal, all-inclusive software architecture does exist yet. Nevertheless, in order to provide an orientation regarding the technical necessities for software architectures and engines in serious game development, this article discusses the following content: First, the specific needs of serious game development in contrast to the needs of game development and software development are identified. The following section on architectures introduces basic game architectures as well as distributed architectures, and it discusses approaches and standards for interoperability. Especially schemes for learning objects and serious games metadata can be considered as essential principles of interoperable architectures. The section about game engines presents a general overview and lists exemplary game engines. Afterwards, a selection of open research questions is followed by summarizing conclusion and further recommended readings.

2 Requirements to Software Development

Although serious games consist of a wide range of manifestations regarding purposes, game mechanisms and technical implementation, there may be a common core of prevalent requirements and characteristics of appropriate architectures and engines. In a first approach to gather further hints, we have a look at the development process. As systematic research about serious game development is rare, we have included game development in our literature review as well. Generally, serious games are a subset of digital games, which are a significant part and driving factor of the creative industries.

2.1 Game Development

It is common practice to describe game development by comparison to software development. This approach is followed by Murphy-Hill et al. [9]. They state that software development for digital games is not a well-researched topic yet. However, they identified tendencies and first important findings in the results of their survey: in game development, the requirements to the product are more unclear compared to conventional software development. Therefore, requirements often are subject to change during the development process. This finding is backed by the complex, non-deterministic and non-linear but iterative process of game design in order to develop working, fun-creating game mechanics. That is, if implemented game mechanics do not seem to work, changes have to be made in order to achieve the development goal [10]. Such a process requires agile development methods, which are in fact more prevalent in game development projects. Murphy-Hill et al. [9] consider the ability to communicate with non-engineers or domain experts as a key-skill of game developers and coin the term of "software cowboys" as one archetype of a successful game developer. Accordingly, Hagan et al. [11] point out, that the combination of people from different disciplines imposes specific importance on communication and team building: "effective collaboration requires a team that respects each other's contributions, communicates frequently

and shares a similar conceptual model of the product and goals". Cooper and Scacchi [12] even underline that game development is a broad and comprising field, which leads to developers being narrowly skilled in probably only one game genre. For the field of online game development Morgan [13] postulates that "the commercial success of online gaming [..] has masked the technological inadequacies". He mentions missing standards for content sharing and non-given interoperability between games (Morgado [14] extends this deficit to virtual worlds) and unavailable model-driven development procedures. The proliferation of Minecraft [15] in educational and research contexts [16, 17] illustrates, that content sharing standards and interoperability can facilitate the use of games: gaming scenarios from a broad range of versatile contexts can easily be provided and deployed by the usage of a standardized platform.

Ampatzoglou and Stamelos [18] conducted a systematic literature review about research topics in game engineering. According to their findings, software development for games recently has become a more vital research field increasing year by year at a disproportionately higher rate compared to software engineering research in general. The thesis of a game development as an emerging and progressing field is supported by Prakash et al. [19]. Most prevalent topics are *requirements and specification*, *management* and *coding tools and techniques*. *Software architecture* and *software reuse* are among the less focused topics. Ampatzoglou and Stamelos [18] cite McShaffry [20], who found diverse criteria to "differentiate game software engineering from classical software engineering", and they conclude that games have a shorter-lifecycle as well as a shorter development period. Product maintenance consists mainly of bug fixing; therefore, no revenues are generated from maintenance releases. Nevertheless, sequels and extensions are considered as a kind of maintenance releases, which create revenues.

Another result of Murphy-Hill et al. [9] is that game development relies often on in-house tools in contrast to standard software. However, Wang and Nordmark [21] identify integration of third party components as one tendency in game development.

Blow [22] uses case studies to describe that game development becomes more complex as technical options progress. Kanode and Haddad [23] identify the combination of multiple and diverse kinds of assets like graphics, videos, code, sound effects as one challenge of game development in contrast to software development. Hagan et al. [11] add that innovation and speed to market are vital in game development.

2.2 Serious Game Development

Serious game development has – compared to plain game development – the additional burden of integrating the "serious" element into the game. This demand complicates the development process further as it removes degrees of freedom from the design process (see chapter "Processes and Models for Serious Game Design & Development" in this volume) in order to achieve a seamless integration of content and game [24].

Among the differences to conventional game development is that the hardware requirements should be rather low [3]. For example, educational application settings of serious games are often connected to schools where not always the most recent hardware is available. In addition, target systems and target groups may be more heterogeneous. The application setting of a serious game may comprise target groups that are

not necessarily gamers. As the example of Social Network Games (SNGs) shows (cf. e.g. [25, 26] and chapter "Social Network Games"), hereby further requirements are imposed on the user interface design beyond digital game standards. Another characteristic of serious games is the necessity of more accurate simulation models. In conventional digital games, simulation models are optimized for entertainment purposes, in most serious games the simulation models are a significant part of the serious content and have to be close to reality.

Testing of serious games not only comprises play tests (in order to ensure fun creating game mechanics). Furthermore, validation steps must be included to test if the actual purpose of the game can be fulfilled. For this reason, monitoring functionality has to be integrated. One approach to accomplish this is the employment of third party tools through the support of standardized interfaces. The possibility to conduct *Learning Analytics* has become an impacting side-goal in serious game design [27, 28].

These additional requirements to design and implementation of serious games are often diametrically opposed to low budgets, and it is considered as one of the main contradiction in serious game development [29]. There are only few cases with exceptional high budgets, e.g. serious games for military training. The most prevalent attempt to face this dilemma is to reduce technical complexity of serious games [30, 31]. Authoring tools are a relevant contribution in this category. Lester et al. [32] define a list of requirement for the specific case of authoring tools for pedagogical agents. They request familiar user interface paradigms with standard editing features, support for author collaboration and rapid iteration and testing. In addition, different levels of experience should be accommodated as well as automation of complex and tedious tasks. Usage should be eased by templates and tutorials [32].

Between developers of commercial entertainment games it is controversially discussed, if in-house game engines or third-party game engines are preferable [33]. Whereas the creation of an own engine can free game developers from vendor-dependency, serious game development typically does not have the necessary budgets to develop an in-house game engine. Thus, this make-or-by-decision commonly is taken away from serious game development in favor of freely available or affordable engines.

In conclusion, serious game development is a complex task in a technically fast moving environment. Whereas commercial games can just focus on achieving a great portion of fun, the development of serious games also has to adhere to the integration of the "serious" goal. In addition to this increased difficulty, technological progress also boosts the capabilities of commercial entertainment games that can be considered as a benchmark for attractiveness of serious games. This tremendously complex task is faced by commonly low budgets. The lack of resources can be compensated at least partially by functionality and efficiency of the underlying software, tools and their compositions, i.e. architectures.

3 Architectures

In its basic form "software architecture deals with abstraction, decomposition and composition, and style and aesthetics" as stated by Kruchten in his 4+1 view model of architecture [34], which is depicted in Fig. 1.

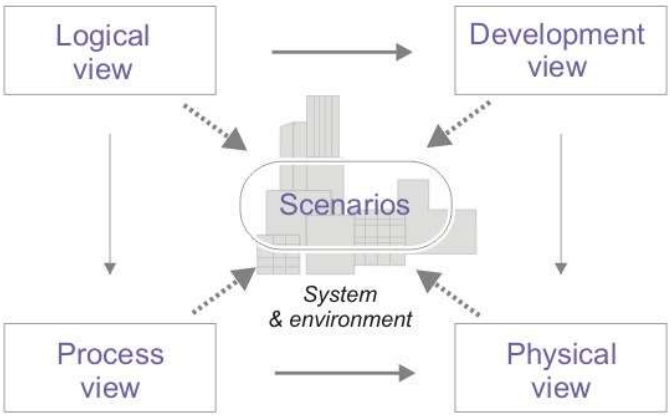

Fig. 1. 4+1 view model of architecture by Kruchten [34]

Kruchten [34] suggests four views of a software architecture to describe its model: (I) a logical view illustrates the object model, (II) a process view deals with concurrency and synchronization issues, (III) a physical view describes the "mapping of the software onto the hardware and reflects its distributed aspects" and (IV) a development view represents the software's static organization in its development environment". Bass et al. [35] add the concepts of components, their interfaces and their interrelated compositions.

The notion of software architecture as a composition of components becomes especially apparent in depictions of game engines, as shown in Fig. 2. Self-contained components with well-defined interfaces are arranged to build a game engine. This representation suggests that the views I and II of Kruchten's 4+1 model [34] are handed by the game engine's architecture: object models have to follow the game engines' technical conventions and within their game loop they frame the handling of synchronization and concurrency issues. The views III and IV are applicable to game architecture itself: the distribution of components among the involved hardware is handled by the physical model. The development model describes the integrated components and libraries. Therefore, we differentiate between the architecture of a *game engine* and the architecture of a *game*. While we deal with the latter in this section, the first is handled in section 4.

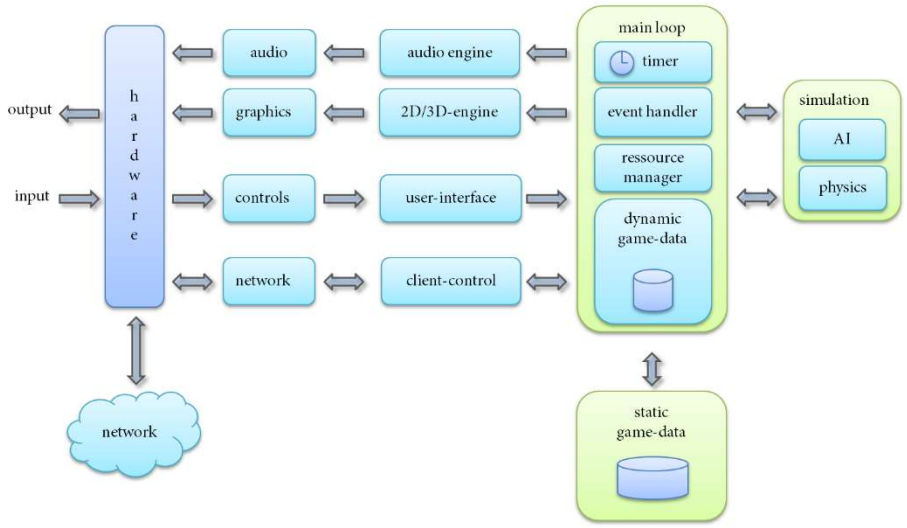

Fig. 2. Game Engine: Software architecture as a composition of components and interfaces [36]

3.1 Game Architectures and its Typical Components

Apart from game engines (software frameworks that provide basic services for game development in order to enable an efficient development process – cf. section 4), there are further components, which are typically included in architectures for (serious) games. Hereafter we look at typical examples of serious games and describe their architectures, focusing on the aspects of the physical and the development view; each of the described exemplars illustrates typical facets. We use the platform categorization from Connolly et al. [2] which comprises *Mobile*, *Internet*, *PC* and *Console*. Non-digital games are excluded, because they do not require a software infrastructure. The category *Virtual World* is disregarded, as the results of Connolly et al. do not indicate significance in the past for the field of serious games (which probably may change with the raise of head-mounted virtual reality displays as *Oculus Rift* [37]). From a technical point of view, this distinction becomes - at least partially – obsolete. game development environments, which support multiple (target) platforms, are evolving (e.g. Unity [38]). However, a complete unification for all platforms will not take place: there is a continuous stream of emerging hardware devices, which facilitate new forms of gaming. Recent developments in this context are, for example, mobile devices, virtual reality displays or sensors. The emergence of new hardware devices leads to further developments of new architectures.

This section continues with a short overview on exemplary architectures for prevalent platforms. It shortly discusses sensors as architecture components and concludes with a paragraph about common components.

PC. *Mobility* is a city-builder game with the focus on traffic simulation [39, 40]. It has been released in 2000 and has been developed with the support of a German industrial and governmental consortium that contributed a budget of approximately € 500.000. Developed as an educational game and installed a million times, it also received popularity as an entertainment game [41]. Its architecture is simple: As a PC game it is installed on a Microsoft Windows-based computer. It does not require an internet connection. Regarding the development view, it is remarkable that it was developed without a dedicated game engine. The entire game has been created from scratch using a C++ programming environment. The monolithic simulation model has been defined beforehand; it comprises 112 relevant factors.

Minecraft [15] is an example of a widespread PC based game, which is used in educational contexts. It has been developed in Java and is able to run without a network connection. The multiplayer mode requires a network connection and a server that hosts a common virtual world.

Mobile Games. *JuraShooter StGB* [42] is a learning app for law students. It facilitates multiple response questions (MRQ) for learning and animates the answering process. From a physical view, there are three components: the app itself, a content management system and the *Apple Game Center* as a rudimentary assessment administration tool (see Fig. 3).

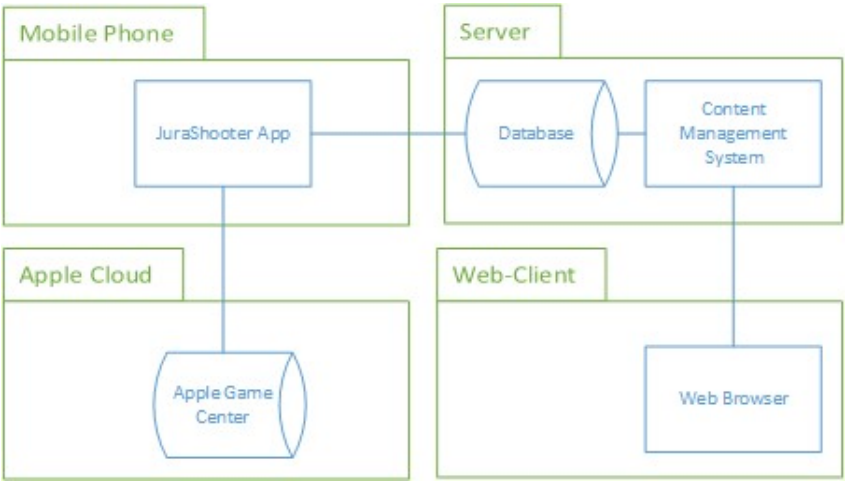

Fig. 3. Physical view of JuraShooter StGB

The JuraShooter StGB has been developed using *Cocos2d* [43], an open source software development framework. The software structure of JuraShooter StGB is based on a framework called *LernShooter*: by supplying another set of graphics, video, sound effects (SFX) and content in the form of MRQs a new app can easily be created. This

has been done various times (e.g. *KanalrattenShooter* [8]). Fig. 4 depicts the development view of a new LernShooter app. Its content will be supplied by the content management system (see Fig. 3).

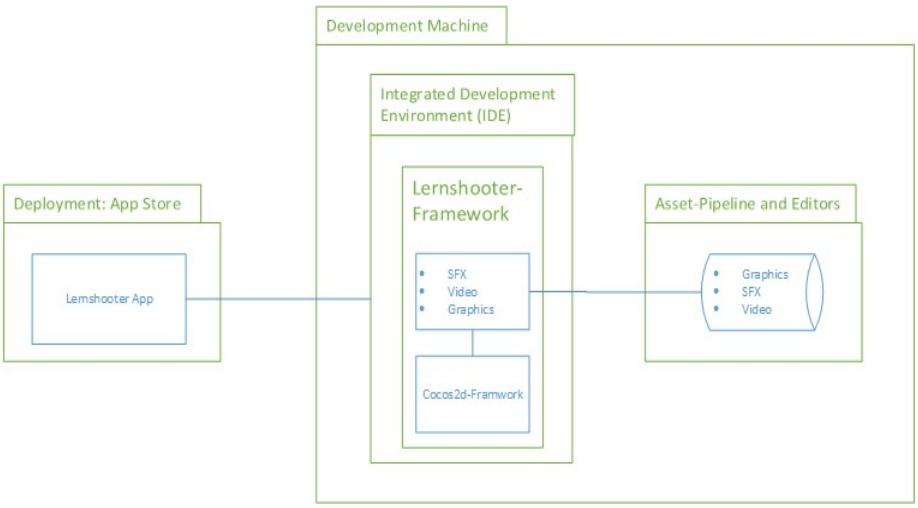

Fig. 4. Development view of the LernShooter architecture

Web-based Games. Web-based Games are operable in a web browser. *Energetika* [44] is an example for a web-based serious game. It received attention, as it has been awarded the *Deutscher Computerspielpreis* (the most important award for digital games in Germany) in the category *Serious Game* in 2011. In that year the nuclear disaster of Fukushima led to a turnaround in Germany's energy policy. Energetika helps to illustrate characteristics and consequences of different energy sources. It is a simulation game about the energy supply for industrial nations, raising issues like carbon dioxide emissions and radioactive waste repositories. Its technical base is the proprietary, Adobe Flash-based framework *Epigene* [45]. The implemented simulation model is comprehensive and self-contained. It is not intended for subsequent extension. The attached database stores highscores of different categories. Among the categories are the energy mix and achievements (see Fig. 6). The deployment is managed via dedicated web servers (see Fig. 5).

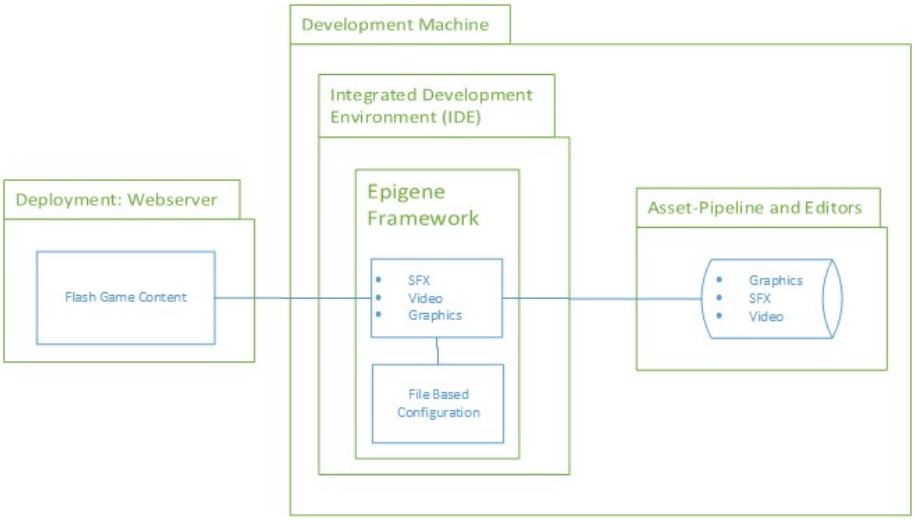

Fig. 5. Development view of a web-based game (Energetika)

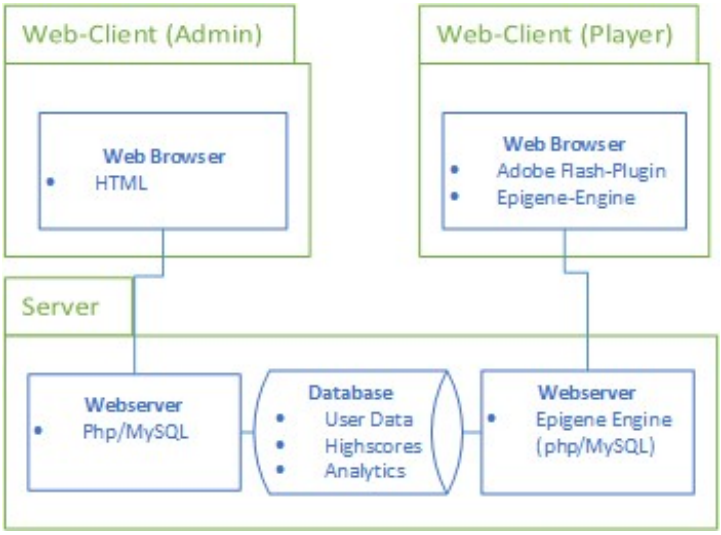

Fig. 6. Physical view of a web-based game (Energetika)

Video Console. Serious games developed for video game consoles are rare. In a review of 129 publications concerning effects of digital games and serious games, Connolly et al. [2] identified 26 that related to console-based entertainment games in serious contexts. However, they could collect only two papers about console games intended for learning purposes and none for other purposes. In general, the deliveries and services of consoles, which are specifically designed hardware devices for gaming, can also be provided by personal computers (PCs). PCs are by far more widespread than consoles

and have no functional limitations compared to consoles. In general, console games do not differ significantly from PC games. However, console games make more use of additional sensory input devices, like the *Nintendo Wii Remote* or the *Microsoft Kinect*. Microsoft Kinect tracks the movements of the players, based on a range-camera. It can facilitate for instance sport simulation games [46]. The Wii Remote achieves a similar effect by using a combination of accelerometer and optical sensors.

Sensors as Architecture Components. As consoles and PCs are by and large functionally equivalent as deployment platforms, we focus on the aspect of sensors as components of serious games. An example of a serious game processing sensor data is the *Aiming Game*. It uses sensor-provided biofeedback by means of electroencephalography and electromyography to train emotion regulation [47]. The game *Letterbird* is an example for an exergame and intends to control the load of endurance training. It processes sensor data of the athlete (heartrate) and the training device (ergometer, pedal rate and resistance setting) [48]. In this context, Hardy et al. [49] have developed a *Framework for adaptive Serious Games for Health* (see Fig. 7). Besides pointing to sensors as an essential part of this framework, it is an example of components forming a system architecture. A kind of exergame, following the metaphor of an air puck, with an additional social dimension has been described by Maier et al. [50]. The experimental setup consists of two PCs, each equipped with a *Kinect* sensor and connected via local area network (LAN). Players' performances are balanced by an adaptation component. As a result, impaired patients can match up with non-impaired.

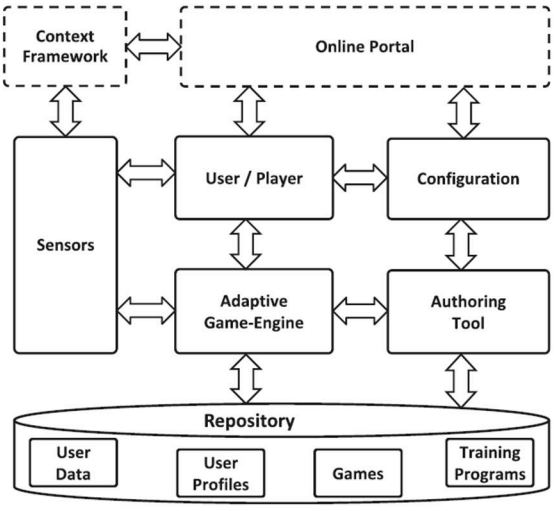

Fig. 7. *Framework for adaptive Serious Games for Health* derived from Hardy et al. [49]

In general, the integration of sensors opens further application areas for serious games. Among these are exergames, Games for Health (see chapter *Games for Health* in this volume), Pervasive Games (see chapter *Pervasive Games* in this volume) and embodied interaction (see chapter *Embodied Interaction in Play* in this volume). Regarding the architecture, it expands the physical view with hardware-based sensors and probably further game engine components. The development view is extended with drivers and programming interfaces.

Common components. Summarizing the previous examples, a game engine (see section 4) is an important part of the development view of almost any architecture of serious games. Commonly, web-based architectures include servers, which host the game itself and deliver it to the web browser. Additionally, they host authoring tools or content management systems as well as the databases with content, user data, sensory input data, etc. Another component used in professional web-based game development is a client-server-middleware handling request management. *SmartFoxServer* [51] is an example for such a middleware. Furthermore, to select an appropriate client/server model is crucial in case of highly frequented virtual worlds [14, 52].

3.2 Distributed Architectures

Distributed software architectures are widely applied to various kinds of application areas where software components are not just located on a single computer but on networked computer systems. Advantages of distributed architectures like shared use of resources (assets, code, logic blocks, etc.), openness, parallelism, scalability or transparency also apply for games. Examples of distributed software architectures are client-server, peer-to-peer or *n*-tier architectures (web applications are a prominent example for the latter). For further reference, Coulouris et al. [53] give a detailed overview of distributed systems, including the theoretical foundation of distributed algorithms. Regarding games, a prominent example of a distributed system are massively multiplayer online games (MMOG) where hundreds of players are interconnected over the internet and play the same game instance in the same game world [54, 55]. Whereas first games have been completely monolithic, today's games are often modular and dynamic. They can load new game content, game states or mechanics from the internet and incorporate them at runtime. Distributed architectures with interoperable data models are needed to realize such games. In the broader field of distributed virtual reality environments and distributed simulations several architectures have been proposed, notably MASSIVE [56], DIVE [57], and the High Level Architecture (HLA) or its predecessor Distributed Interactive Simulation (DIS).

Whereas distributed game architectures are commonly used in popular commercial games, they are rarely applied to serious games. Carvalho et al. [58] introduce the concept of a service-oriented architecture (SOA) as an architectural model for serious games. In this model, the game components are distributed to various servers offering their services through web interfaces. The advantages are up-to-date components, au-

tomated detection of services and exchangeability of service providers in case of standardized interfaces. Furthermore, it removes special hardware requirements from clients as web browser access enables ubiquitous accessibility.

A distributed, multi-agent system has been presented by van Oijen et al. [59] in their CIGA middleware for intelligent virtual agents (IVAs). CIGA is a software architecture to connect multi-agent systems to game engines using ontologies as a design contract. The CIGA middleware negotiates between the physical layer of game engines and the cognitive layer of multi-agent systems (IVA).

Jepp et al. [60] describe an agent-based architecture for modular serious games. The framework strives to provide serious games with believable, emotional agents to help players learn skills and evaluate their performance. It is part of the TARGET platform helping learners to train competencies in game scenarios. The framework is interlinked with a game engine, a dialogue system and a narrative engine via a so-called translation engine handling synchronization and communication.

A distributed architecture for testing, training and simulation in the military domain is TENA by the U.S. Department of Defense (DoD) [61]. This extensive architecture focuses on interoperability for military test and training systems. At its core, the TNA middleware interconnects various TENA applications and tools for the management, monitoring, analysis, etc. of military assets. Via a gateway service, they can be linked with other DIS or HLA conformant simulators that provide real sensor data or data from live, virtual and constructive simulations (LVC) for the TENA environment.

Peirce et al. [62] present the ALIGN architecture to enable the adaptability of serious games in a minimal-invasively fashion. The ALIGN system architecture decouples the adaptation logic from the actual game without mitigating the game play. It is divided into four conceptual processes: the accumulation of context information about the game state; the interpretation of the current learner state; the search for matching intervention constraints; and a recommendation engine, which applies adaptation rules to the game. ALIGN is not included in the actual game but communicates with attached game engines via TCP/IP. It has been applied in the educational adventure game of the ELEKTRA project [62].

3.3 Data Models and Interoperability

While the software architecture specifies the structure of a software system and how data flows through it, the technical specification of the data itself (the schema) is also part of the overall system specification and the specification of the scenarios. This section focuses on the data schemas for interoperability. In the domain of Modeling and Simulation (M&S) exist various standards to describe virtual environments, e.g. the *Simulation Reference Markup Language* (SRML) [63, 64] or state-machine models like *State-Chart XML* (SCXML) [65].

Standards for technical data representation enable systems to be interoperable, that is to effectively exchange data and information. In sustainable IT environments the interoperability of data and processes is one of the core aspects for efficiency, responsiveness and cost reduction. This applies to serious games as well: Only when data schemas are interoperable with other serious games, true data exchange is effectively

possible (e.g. data exchange on usage activity or content). Stãnescu et al. [66] give an overview of the interoperability of serious games. They propose a *Serious Games Multidimensional Framework* (SG-MIF) to consider different levels of interoperability, regarding serious games components, their ecosystem and how to handle topics following the use of serious games [66].

For learning management systems (LMS) various data models and exchange formats have been proposed, notably the IEEE *Learning Object Metadata* LOM) [67] or the exchange format *Shareable Content Object Reference Model* (SCORM) [68]. The IEEE-LOM is a base schema to annotate learning resources with metadata. Annotated learning resources are called Learning Objects (LO). LOM was developed to facilitate the search, acquisition, exchange and use of LOs. It allows the specification of new application profiles with mixed element sets and references to other vocabularies. Whereas LOM specifies a schema for metadata annotation, SCORM makes use of it in its own LOM application profile: It provides a collection of standards for the communication and data exchange of LOs. SCORM includes a specification about packaging LOs for interchange between different LMSs. What sounds good in theory has its pitfalls in reality. A simple exchange of learning resources via SCORM proved to be difficult and far from "plug & learn". Simple, static data can be exchanged but dynamic content or learning settings cannot, because of the different learning concepts of the LMSs. For Intelligent Tutoring Systems (ITS) and adaptive learning systems LOM is a key model for repurposing content or aligning it along adaptive learning pathways [69, 70]. The effective reusability and repurposing of learning objects offer the possibility of an efficient game development process. It enables the computer to personalize and adapt games to individual users by, e.g. rearranging content. The reusability of learning objects has already been demonstrated between web-based LMS and game-based LMS [71]. The data exchange between games and LMS has been shown in the <e-Adventure> platform by Torrente et al. [72]. It is a set of platforms for developing inexpensive, educational games, including an API for tracking and assessment. The API can transfer results of the game play to a learning management system (LMS), and thus, students' performances can be monitored and aggregated. Similarly, the API implemented by the <e-Adventure> platform follows the SCORM specifications [68], which eases integrations with LMS servers that adhere to this standard.

Further research has to be done to establish a commonly accepted metadata schema for serious games. LOM as an already accepted base schema, provides the foundation for such a schema. A taxonomy of educational games that is compatible to IEEE-LOM is presented by Silva *et* al. [73].

El Borji and Khadi [74] present an application profile of the IEEE LOM as the so-called *SG-LOM* for serious games (see Fig. 8). The intention is to use serious games as learning resources that are integrated into existing LMS. This metadata schema allows to exchange tracking and assessment data between serious games and LMS.

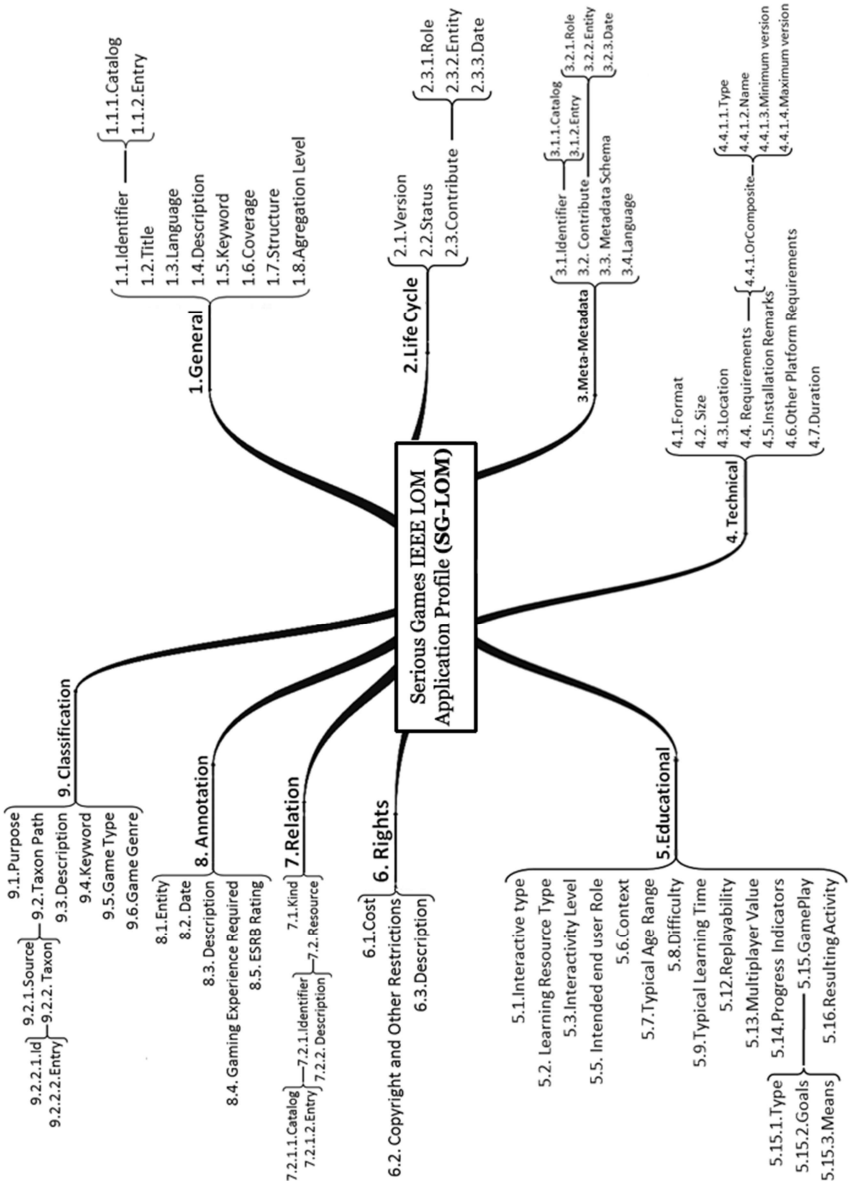

Fig. 8. Learning Object Model (LOM) application profile for Serious Games [74]

4 Engines

This section gives a short introduction to game engines in general. It describes the function of common core components, lists examples of popular game engines and presents methods how to categorize them.

4.1 Overview

In software engineering, the term *engine* is reserved for a self-contained software framework that processes input data into output data (input-process-output model). The applied processes as well as the input and output data can be described formally, e.g. using formal modeling standards like UML2 [75], State-Chart XML [65], etc., or using XML schema to define the data models. There are different kinds of software engines, e.g. simulation engines, search engines, inference engines, etc. An engine is a part of a software system; thus, a game engine is usually an essential part of game software. Engines offer services that free programmers from developing low-level algorithms, like algorithms for computer graphics and rendering or for user-adaptive non-player character (NPC) behavior, thus engines contribute to efficient game development. For example, a game engine often provides the service of visualizing 2D or 3D worlds. Rucker [76] describes the game development framework *Pop*, which has been developed for educational purposes. It documents basic requirements, processes in development and the usage of an elementary game engine.

In his description of the evolution of game engines, Gregory "reserve[s] the term 'game engine' for software that is extensible and can be used as the foundation for many different games without major modification" [77]. In a short definition Anderson et al. [78] identify reusable software components as game engines. They point out that different game genres require different game engines. Gregory supports this statement: a game genre often has specific requirements on a game engine. However, he admits that borders have blurred with the technical progress of game engines. Nowadays there are game engines that are capable of being applied to various genres. Resulting games may not achieve the quality of commercial games, but they deliver astonishingly acceptable player experiences. [77]

Often a game engine provides multiple aspects that are included in game play. A set of aspects of digital games, which commonly are handled by game engines, are shown in Fig. 2. This figure also includes the main structure (i.e. architecture) of a typical game engine. A main control loop handles events and calls and coordinates various components of the engine. Often these components themselves are called "engines" as they handle a single aspect of the game engine. Therefore, the audio engine is responsible for providing an acoustical environment in the game.

Authoring environments for games (e.g. StoryTec, G-Flash, <e-Adventure> and SeGAE [72, 79–81]) differ from game engines: the game designer has to provide content to create a self-contained game. In contrast to an authoring environment a game engine provides just the base for a digital game – game designers have to design the game and programmers have to transform it into a working game using the game engine's services.

Game engines provide very powerful concepts. One challenge in game design is to use these concepts purposefully to provide immersive and engaging games. Usage of features which are not appropriately integrated in the game design concept (e.g. the narrative) may cause an unwanted distraction and may lead to "breaking the magic cir-

cle" [82]. In his discussion of physics in a game, Gregory points out that non-purposefully used physics features may distract the player from the intended game experience [67].

4.2 Selected Components

Although most functionalities of a game engine are provided unrecognized as transparent services to the game developers, it is reasonable to introduce some components in detail. This may be useful as they have to be developed or adapted to serious games (like personalization or adaptation); or the knowledge about their principles helps to understand the development process (as in the case of 2D/3D engines or physics engines).

2D/3D-Engine. The *graphics engine* is a component on the view- or presentation-layer and handles all rendering of graphics, text and symbols (i.e. all visuals). Graphics are an important part of a game. Graphical effects and quality are often considered as to play a significant role in the playing experience. Almost every game engine includes a graphics engine. Functionalities of a graphics engine primarily comprise visualization of geometric objects and handling of textures. Additionally, visual effects like shading, transparency and reflections also belong to the capabilities of a graphics engine. Rendering is the process of creating an image from the 2D or 3D model. An introduction to the principles of a graphics engine is given in [20].

Artificial Intelligence (AI). In order to provide challenging scenarios for players, AI is introduced into games. A common purpose of AI is to NPC. To challenge the player, the AI plans the reactions of the NPCs in real-time, according to the player's actions. The implementation of AI is demonstrated for example in the first-person-shooter *F.E.A.R.* [83] or in other fight games [84]. Recently, a main goal in the field of AI focusses on the design of more reasonable and believable AI behavior. Besides these extensions, Yannakakis, discusses three further areas of AI in games [85]: (1) *Player experience modeling* (PEM) adopts the game environment to the player's capabilities and preferences. It can provide an individualized and therefore intriguing game play. (2) *Procedural content generation* (PCG) deals with the automated generation of game elements, e.g. creation of new game levels. As content has not to be created manually, it reduces development costs. Furthermore, the created content can be adapted to the player (see chapter *Content generation for Serious Games* in this volume). Additionally, Yannakakis points out the capability of PCG to contribute to design solutions beyond human imagination. The last application area discussed is (3) *Massive-scale Game Data Mining.* It is used to optimize a game "around the player". Conventional algorithms can no longer handle the amount of available data ("Big Data"). This has led to the facilitation of data mining algorithms in games: Collaborative filtering, for example, searches for recurring patterns in interaction data of other human users. The data is matched to the user model of the current user to provide more human-like behaviors.

Commonly a game engine does not support the complete range of AI engines. Instead, AI engines are available as plugins. The *Unity* game engine for example [38] supports a limited set of AI technologies including algorithms for finite state machines, pathfinding and navigation. Additional plugins can be found in Unity's distribution platform *Asset Store*. For instance *RAIN for Unity* adds behavior trees to control the behavior, navigation, motion or animation [86].

Physics. Virtual worlds in games often include simulations of physics. Some genres, like shooter games, depend on it. In game development classical laws of mechanics are subsumed under the term *physics* [87]. Based on the concept of gravity (an object falls down) further phenomena like rigid bodies dynamics (an object is not deformed when external forces impact it) and collision detection (two objects partly overlap during their movement trajectories) are implemented by *physics engines*. Although a physics engine is often implemented as a separate software component, most game engines include them. Other game engines facilitate third party engines. For example, *PhysX*, an engine by NVIDIA that outsources calculations to the graphical processing unit (GPU) is used by *Unity* [88]. Physics engines allow the programmer to model a virtual world by means of configuration to avoid manual programming. As physics calculations are often computationally demanding, offline pre-calculations of movement sequences can be an appropriate optimization strategy [77].

4.3 Game Engine Examples

In literature, a few selection processes of game engines for serious contexts are documented. As already mentioned in section 2.2, a relevant criterion is the ease of use. It should enable even domain experts, who are not experienced game developers, to create serious games and related tools [32]. Cowan and Kapralos [89] emphasize available budgets as a main difference between commercial and serious game development. They conducted a literature research to identify prevalent game engines and frameworks in the context of serious games. Their results indicate that mostly game engines produced for commercial entertainment settings are used for developing serious games. The ten top-most named game engines or frameworks identified (in combination with the keywords *simulator*, *serious game* and *educational game*) are, ordered by decreasing frequency: *Second Life*, *Unity* [38], *Unreal* [90], *Flash*, *XNA*, *Torque*, *OGRE*, *GameMaker* [91], *StoryTec* and *OLIVE*.

In 2007 Marks et al. [92] describe the selection of a game engine as a well-tested foundation for simulated surgical training. Their main selection criteria have been inexpensiveness and popularity of the game engines. The final list of candidates consisted of *Unreal Engine*, *id Tech 4* and *Source Engine*. Examination criteria have been the availability of editing features, integration of further content and support of multiple users of the simulator.

As resources for serious game development are often are limited, the results of Rocha et al. [93] are relevant: In 2010 they evaluated the market of open source 3D game

engines. Their sub-criteria have been: (a) recent stable version, (b) source code is available, (c) support for both operating systems, *Windows* and *Linux,* (d) not restricted to a specific game genre and (e) documentation is available and a lively community of users. They found the following game engines and tool kits: *Blender Game Engine* [94], *Crystal Space* [95], *Delta3D* [96], *Irrlicht* [97], *jMonkey Engine* [98], *Ogre3D* [99], *OpenSceneGraph* [100] and *Panda3D* [101]. These engines are discussed regarding the categories of graphics and non-graphics features, their development support and their organizational and technical maturity.

Petridis et al. [102] propose a framework for selecting game engines for serious games in high fidelity applications. Fidelity in their context refers to audiovisual and functional fidelity. Among the applied criteria are **audiovisual fidelity** (to enable immersion), **functional fidelity** (to support learning goals), **composability** (which relates to the reuse of existing components), **accessibility** (as serious games have to support inexperienced players, serious games should not require knowledge about standard game operations), **networking** (to enable multiple users to create a social context), and **heterogeneity** (i.e. multiplatform support). In order to validate this framework, they reduced a comprehensive list of game engines according to the criteria of wide usage, availability, modularity and innovative features. Finally they evaluated four game engines: *CryEngine* [103], Valve's *Source Engine*[1] [104], *Unreal* [90], and *Unity* [38]. These engines can be considered as a good match to the provided framework. However, it needs further refinement as the field progresses at a fast pace.

Westhoven and Alexander [105] proposed a further methodological attempt to structure the selection process of game engines. They extended the framework of Petridis et al. [102] with aspects mentioned by Marks et al. [92] and Sarhan [106] (cf. Table 1) to select a game engine for a Virtual Reality (VR) application. They divided their criteria catalog into three categories: *software-related* criteria, which focus on the technical capabilities of the game engine; *development-related* criteria, which characterize requirements for the development process; and finally *acquisiton-related* criteria. The latter categorie summarizes criteria which impact the provision of a game engine in a concrete development setting. In their case study they demonstrate that the proposed selection criteria have to be taylored to specific requirements. They evaluated *Unity* and *CryENGINE* as potential development platforms for their VR application.

Table 1. Criteria for selecting game engines as proposed by Westhofen and Alexander [105]

Software	Development	Acquisition
Audiovisual display • Rendering • Animation • Sound • Streaming	Accessibility • Documentation • Support • Code Access • Introduction Effort	Accessibility • Licensing • Cost • System Requirements
Functional display • Scripting		

[1] *Source Engine* has been discontinued in 2014. Its successor, *Source Engine 2,* has been announced [125].

• Supported AI • Physics Engine • Event Handling		
Combinability • Component export/import • Development tools		
Networking • Client-Server • Peer-to-Peer		
Heterogenity • Multi-platform support		

Besides commercial game engines, there are specific educational game engines. These are mainly applied in contexts where the process of creating games itself is an educational measure. *GameMaker* [91] is a typical representative of this group. A more complete list of game making tools is maintained on Google Docs [107].

Another kind of game engines is specialized on browser games. These game engines mostly use HTML5 technology or provide browser-plugin support. *GameMaker,* for example, provides export to HTML5 besides other formats.

Seldom game engines are created from scratch by a game developer studio. The reasons for own developments are to support a specific genre and to reuse results for other games – as it has been done in the case of *CryEngine*, *SourceEngine* and *Unreal*. An remarkable attempt has been undertaken for SimCity 5 [108], because not only graphics but also the modeling of the simulation itself can be adapted: the underlying *GlassBox* engine, which has not been disclosed to the public, uses agent-based simulation and adheres to the *What You See Is What You Simulate Principle* [109, 110].

5 Research Questions (starting points for PhDs)

The most prominent research questions mostly concern the **efficient game development process**. As stated in section 2, serious games require a huge development effort by design. Because of their multidisciplinary nature, their development could in fact cost more than business applications. Concepts for easy transferable – or even universally applicable – architectures, as well as supporting and flexible game engines are needed.

Before thinking about universally applicable architectures first steps towards commonly accepted **architecture blueprints** have to be devised. In software engineering, architecture blueprints are a proven tool in the efficient development of good and sustainable software products [111]. Standardization of architectures and data schemas could provide the community with interoperable data models and facilitate the data exchange between applications.

The integration of **emerging end user interfaces** into serious games is another reoccurring topic of research [14]. Such technologies could include cloud-based rendering (e.g. OTOY [112]) and virtual (e.g. Oculus Rift [37]) and augmented reality devices (e.g. Google Glasses). Data from other sources (e.g. Internet of Things) and sensors (e.g. eye tracking devices, Xbox Kinect, etc.) could enable new fields of application of serious games.

In general, **arising gaming technologies** have to reviewed regarding to their facilitation of serious gaming (e.g. cloud gaming or computation offloading [113]).

Domain-specific game engines can increase the efficiency of development. An example is SimCity's *GlassBox* engine. It eases development by visualizing the simulation processes specific to city building games. The same principle can be used to **integrate further domain specific features** into game engines or integrated development environments (IDEs). A candidate could be the integration of learning analytics components and further supporting tools for educational games.

Model driven development decreases development efforts. Providing tools to generate games from models would ease the development process and probably lower the requirements to the technical knowledge of game designers and developers. **Authoring tools** ease game development in a similar way, but probably have a limited variability of resulting products.

Interoperability and data exchange has been identified as crucial for a widespread usage of serious game. However, it is not applied in practice at a reasonable rate. Among the reasons are technical limitations, which have to be eliminated.

6 Summary & Outlook

Serious game development is a highly complex and demanding process that relies on expert knowledge and software development experience. Currently available game engines and common architectures support the creation of decent serious games. However, there is potential for improvements.

There are some main challenges in serious game development: heterogeneous teams and a perpetual tendency of changing requirements. These issues are by far more specific to game development than to conventional software development. Commercial digital games provide a benchmark in terms of fun and entertainment. The need for "serious" content complicates game design. Development budgets are often comparatively small. Additional components like sensors and data have to be included in game architectures. Technical possibilities change at a fast pace.

The main contribution of architectures and game engines is to improve the efficiency of the development process. The quality of serious games in terms of resulting engagement level has to follow those of pure entertainment games. For this reason, there will always be a pressure to enhance the technical foundation of serious games. This technical foundation includes tools specific to the requirements of serious games (e.g. the integration of learning analytics in educational games).

Although in recent years the technical foundation has improved significantly, there is still great potential to enhance the efficiency of serious game development. Established principles of conventional software development have not yet been applied to serious games. Therefore, besides developing game specific technologies and algorithms, conventional software development can be used as a pool of inspiration and ideas to streamline serious game development.

7 Further Readings and Resources

7.1 Books

- **Gregory, J. *Game Engine Architecture* (2014).** A basic reading about the internals of game engines. It is one of the standard references for services offered by game engines and therein applied algorithms and principles [77]
- **Cooper, K. M. L., & Scacchi, W. *Computer Games and Software Engineering* (2015).** A recent collection of academic articles giving an overview about difficulties in software engineering for digital games. It serves as a starting point for a more theoretical approach to the topic.[114]
- **McShaffry, M., Graham, D.: *Game Coding Complete* (2012).** The fourth edition of a standard work in game development. It gives an overview of the (technical) challenges in game development and delivers recipes to master them. [20]
- **Hocking, J. *Unity in Action: Multiplatform Game Development in C# with Unity 5* (2015).** A well-written recent introduction into the currently leading game development tool. This book is an excellent resource in case a concrete initial implementation, based on the widespread game engine Unity, is intended. [115]
- **Nystrom, B. *Game Programming Patterns* (2014).** A well-received book about the principles of game development. It covers specific patterns, which solve problems occurring specifically in game development. Thus, this book helps both to understand common game engines and game architectures and to design and implement proprietary ones. [116]

Among further notable books are [54, 55, 117].

7.2 Websites

Gamasutra is a considerable online magazine about commercial digital game development. Among websites about technical aspects of game development are further **gamedev.net** [118] and **AIGameDev** [119]. A Q&A-platform about game development is provided by **StackExchange**[120].

Databases collecting information about game engines can be found on the websites **DevMaster** [121] and **HTML5 Game Engines** [122]

7.3 Conferences

All issues of commercial game development are addressed at the **Game Developers Conference (GDC).** It is the most renowned, mainly non-academic conference about development of digital games.

In the academic sector there are a few conferences dedicated to serious games, which allow discussions about their technical foundations. Among them are the **European Conference on Games Based Learning (EGBL)** and the **Joint Conference on Serious Games (JCSG).** Technical aspects of game development in general are handled for example by **IFIP International Conference on Entertainment Computing (ICEC)** and **Advances in Computer Entertainment Technology (ACE).** Digital games in general are discussed at the renowned **DiGRA Conference** and **Foundations of Digital Games (FDG)**.

7.4 Mailing Lists

The **DiGRA-Mailing List** [123] is highly frequented and discusses all topics of digital games. Another relevant mailing list is maintained by the **IFIP Entertainment Computing Community** [124].

8 References

1. Schmidt, R., Emmerich, K., Schmidt, B.: Applied Games – In Search of a New Definition. In: Entertainment Computing – ICEC 2015 14th International Conference, Trondheim, Norway, Proceedings. pp. 100–111 (2015).
2. Connolly, T.M., Boyle, E. a., MacArthur, E., Hainey, T., Boyle, J.M.: A systematic literature review of empirical evidence on computer games and serious games. Comput. Educ. 59, 661–686 (2012).
3. Michael, D.R., Chen, S.L.: Serious Games: Games That Educate, Train, and Inform. Course Technology, Mason, OH, USA (2005).
4. Djaouti, D., Alvarez, J., Jessel, J.-P.: Classifying serious games: The G/P/S model. In: Felicia, P. (ed.) Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches. pp. 118–136. IGI Global, Hershey, PA, USA (2011).
5. Ratan, R., Ritterfeld, U.: Classifying Serious Games. In: Ritterfeld, U., Cody, M., and Vorderer, P. (eds.) Serious games: Mechanisms and effects. pp. 10–22. Routledge, New York (2009).
6. Streicher, A., Szentes, D., Roller, W.: Scenario Assistant for Complex System Configurations. IADIS Int. J. Comput. Sci. Inf. Syst. 9, 38–52 (2014).
7. Reuter, C., Tregel, T., Mehm, F., Göbel, S., Steinmetz, R.: Rapid Prototyping for Multiplayer Serious Games. In: Busch, C. (ed.) Proceedings of the 8th European Conference on Games Based Learning. pp. 478–486. Vol. 2, Reading (2014).
8. Söbke, H., Chan, E., Buttlar, R. von, Große-Wortmann, J., Londong, J.: Cat King's Metamorphosis - The Reuse of an Educational Game in a Further Technical Domain. In: Göbel, S. and Wiemeyer, J. (eds.) Games for Training, Education, Health and Sports.

pp. 12–22. Springer International Publishing, Darmstadt (2014).
9. Murphy-Hill, E., Thomas Zimmermann, Nagappan, N.: Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development? In: 36th International Conference on Software Engineering (ACM). pp. 1–11 (2014).
10. Fullerton, T.: Game Design Workshop: A Playcentric Approach to Creating Innovative Games. Morgan Kaufmann, Burlington, MA, USA (2008).
11. Hagan, A.O.O., Coleman, G., Connor, R.V.O.: Software Development Processes for Games: A Systematic Literature Review. Proc. 2008 Conf. Futur. Play Res. Play. Share. 425, 182–193 (2014).
12. Cooper, K.M.L., Scacchi, W.: Introducing Computer Games and Software Engineering. In: Cooper, K.M.L. and Scacchi, W. (eds.) Computer games and software engineering. pp. 1–27. Chapman and Hall/CRC, Boca Raton, FL, USA (2015).
13. Morgan, G.: Challenges of Online Game Development: A Review. Simul. Gaming. 40, 688–710 (2009).
14. Morgado, L.: Technology Challenges of Virtual Worlds in Education and Training - Research Directions. Games Virtual Worlds Serious Appl. (VS-GAMES), 2013 5th Int. Conf. 1–5 (2013).
15. Mojang: Minecraft, https://minecraft.net/.
16. Nebel, S., Schneider, S., Rey, G.D.: Mining Learning and Crafting Scientific Experiments: A Literature Review on the Use of Minecraft in Education and Research. J. Educ. Technol. Soc. 19, 355–366 (2016).
17. Petrov, A.: Using Minecraft in Education: A Qualitative Study on Benefits and Challenges of Game-Based Education, https://tspace.library.utoronto.ca/bitstream/-1807/67048/1/Petrov_Anton_201406_MT_MTRP.pdf, (2014).
18. Ampatzoglou, A., Stamelos, I.: Software engineering research for computer games: A systematic review. Inf. Softw. Technol. 52, 888–901 (2010).
19. Prakash, E., Brindle, G., Jones, K., Zhou, S., Chaudhari, N.S., Wong, K.-W.: Advances in Games Technology: Software, Models, and Intelligence. Simul. Gaming. 40, 752–801 (2009).
20. McShaffry, M., Graham, D.: Game Coding Complete. Course Technology, Boston, MA, USA (2012).).
21. Wang, A.I., Nordmark, N.: Software Architectures and the Creative Processes in Game Development. In: Entertainment Computing – ICEC 2015 14th International Conference, Trondheim, Norway, Proceedings. pp. 272–285. Springer International Publishing Switzerland, Cham (2015).
22. Blow, J.: Game Development: Harder than you think. Queue. 1, 28–37 (2004).
23. Kanode, C.M., Haddad, H.M.: Software Engineering Challenges in Game Development. In: Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on. pp. 260–265. IEEE (2009).
24. Habgood, M.P.J., Ainsworth, S.E.: Motivating Children to Learn Effectively: Exploring the Value of Intrinsic Integration in Educational Games. J. Learn. Sci. 20, 169–206 (2011).
25. Fields, T.: Mobile & Social Game Design: Monetization Methods and Mechanics. A K Peters/CRC Press, Boca Raton, FL, USA (2014).

26. Kinder, K.: You have a Farmville gift request":Thesen zum Erfolg von Social Casual Gaming auf Facebook. kommunikation @ gesellschaft. 13, 19 pages (2012).
27. Khalil, M., Ebner, M.: Learning Analytics: Principles and Constraints. In: Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications. pp. 1326–1336. Association for the Advancement of Computing in Education (AACE). (2015).
28. Serrano-Laguna, Á., Torrente, J., Moreno-Ger, P., Fernández-Manjón, B.: Tracing a Little for Big Improvements: Application of Learning Analytics and Videogames for Student Assessment. Procedia Comput. Sci. 15, 203–209 (2012).
29. Torrente, J., Lavín-Mera, P., Moreno-Ger, P., Fernández-Manjón, B.: Coordinating Heterogeneous Game-Based Learning Approaches in Online Learning Environments. In: Pan, Z., Cheok, A.D., Müller, W., and Rhalibi, A. El (eds.) Transactions on Edutainment II. pp. 1–18. Springer-Verlag Berlin Heidelberg, Berlin (2009).
30. Moreno-Ger, P., Torrente, J., Bustamante, J., Fernández-Galaz, C., Fernández-Manjón, B., Comas-Rengifo, M.D.: Application of a low-cost web-based simulation to improve students' practical skills in medical education. Int. J. Med. Inform. 79, 459–67 (2010).
31. Warren, S.J., Jones, G.: Overcoming educational game development costs with lateral innovation: Chalk House, The Door, and Broken Window. J. Appl. Instr. Des. 4, 51–63 (2012).
32. Lester, J., Mott, B., Rowe, J., Taylor, R.: Design Principles for Pedagogical Agent Authoring Tools. In: Sottilare, R.A., Graesser, A.C., Hu, X., and Brawner, K. (eds.) Design Recommendations for Intelligent Tutoring Systems Volume 3 Authoring Tools and Expert Modeling Techniques. pp. 151–160. U.S. Army Research Laboratory, Orlando, FL, USA (2015).
33. Hauser, D.D.: License an engine or create your own. Mak. games. 6, 41–45 (2015).
34. Kruchten, P.B.: The 4+1 View Model of Architecture. IEEE Softw. 12, 42–50 (1995).
35. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison Wesley, Boston, MA, USA (2003).
36. Masuch, M., Abbadi, M., Konert, J., Streicher, A., Söbke, H., Dey, R.: Lecture "Serious Game Technology" Dagstuhl GI Seminar 15283 on Entertainment Computing and Serious Games. (2015).
37. Oculus VR LLC: Oculus, https://www.oculus.com/.
38. Unity Technologies: Unity - Game Engine, https://unity3d.com/.
39. Brannolte, U., Harder, R.J., Kraus, T.J.: Virtual City and Traffic Simulation Game Based on Scientific Models. In: Zupančič, B., Karba, R., and Blažič, S. (eds.) EUROSIM 2007: Proceedings of the 6th EUROSIM Congress on Modelling and Simulation, [9-13 September, 2007, Ljubljana, Slovenia], Volume 1. Argesim (2007)..
40. Glamus GmbH: Mobility - A city in motion!, http://www.mobility-online.de.
41. Schmitz, P.: CD-ROM-Kritik: Mobility. c't Mag. für Comput. 252 (2000).
42. Buttlar, R. von, Kurkowski, S., Schmidt, F.A., Pannicke, D.: Die Jagd nach dem Katzenkönig. In: Kaminski, W. and Lorber, M. (eds.) Gamebased Learning: Clash of Realities 2012. pp. 201–214. Kopäd, München (2012).
43. cocos2d.org: Cocos2d, http://cocos2d.org/, (2008).
44. Takomat GmbH: Energetika, http://www.wir-ernten-was-wir-saeen.de/energiespiel/.
45. Takomat GmbH: Good Games - takomat Games | Neue Lebensformen für Medien,

http://www.takomat-games.com/en/games/good-games.html.
46. Rare: Kinect Sports, (2010).
47. Cederholm, H., Hilborn, O., Lindley, C., Sennersten, C., Eriksson, J.: The Aiming Game: Using a Game with Biofeedback for Training in Emotion Regulation. In: Copier, A., Kennedy, M., and Waern, H. (eds.) DiGRA '11 - Proceedings of the 2011 DiGRA International Conference: Think Design Play. DiGRA/Utrecht School of the Arts (2011).
48. Hoffmann, K., Wiemeyer, J., Hardy, S., Göbel, S.: Personalized Adaptive Control of Training Load in Exergames from a Sport-Scientific Perspective. In: Göbel, S. and Wiemeyer, J. (eds.) Games for Training, Education, Health and Sports. pp. 129–140. Springer International Publishing Switzerland, Cham (2014).
49. Hardy, S., Dutz, T., Wiemeyer, J., Göbel, S., Steinmetz, R.: Framework for personalized and adaptive game-based training programs in health sport. Multimed. Tools Appl. 74, 5289–5311 (2015).
50. Maier, M., Rubio Ballester, B., Duarte, E., Duff, A., Verschure, P.F.M.J.: Social integration of stroke patients through the multiplayer rehabilitation gaming system. In: Göbel, S. and Wiemeyer, J. (eds.) Games for Training, Education, Health and Sports. pp. 100–114. Springer International Publishing Switzerland, Cham (2014).
51. SmartFoxServer, http://www.smartfoxserver.com/.
52. Street, S.: Massively Multiplayer Games Using a Distributed Services Approach. In: Alexander, T. (ed.) Massively Multiplayer Game Development. pp. 233–241. Charles River Media, Boston (2005).
53. Coulouris, G., Dollimore, J., Kindberg, T., Blair, G.: Distributed Systems: Concepts and Design. Pearson, Harlow, Essex (2011).
54. Hall, R., Novak, J.: Game Development Essentials: Online Game Development. Delmar, Clifton Park, NY, USA (2008).
55. Alexander, T. ed: Massively Multiplayer Game Development 2. Charles River Media, Newton Centre, MA, USA (2005).
56. Greenhalgh, C., Benford, S.: MASSIVE: a collaborative virtual environment for teleconferencing. ACM Trans. Comput. Interact. 2, 239–261 (1995).
57. Frécon, E., Stenius, M.: DIVE: a scaleable network architecture for distributed virtual environments. Distrib. Syst. Eng. 5, 91–100 (1998).
58. Carvalho, M.B., Bellotti, F., Berta, R., De Gloria, A., Gazzarata, G., Hu, J., Kickmeier-Rust, M.: A case study on Service-Oriented Architecture for Serious Games. Entertain. Comput. 6, 1–10 (2015).
59. van Oijen, J., Vanhée, L., Dignum, F.: CIGA: A Middleware for Intelligent Agents in Virtual Environments. In: Proceedings of the 2011 International Conference on Agents for Educational Games and Simulations. pp. 22–37. Springer-Verlag, Berlin, Heidelberg (2012).
60. Jepp, P., Fradinho, M., Pereira, J.M.: An Agent Framework For a Modular Serious Game. In: 2nd International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2010. pp. 19–26 (2010).
61. Noseworthy, J.R.: The Test and Training Enabling Architecture (TENA) Supporting the Decentralized Development of Distributed Applications and LVC Simulations. In: Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th

IEEE/ACM International Symposium on. pp. 259–268 (2008).
62. Peirce, N., Conlan, O., Wade, V.: Adaptive Educational Games: Providing Non-invasive Personalised Learning Experiences. In: 2008 Second IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning. pp. 28–35. IEEE (2008).
63. Reichenthal, S.W.: The Simulation Reference Markup Language (SRML): a foundation for representing BOMs and supporting reuse. Proc. Fall 2002 Simul. Interoperability Work. 1, 285–290 (2002).
64. Reichenthal, S.W.: SRML - Simulation Reference Markup Language, https://www.w3.org/TR/SRML/.
65. Barnett, J., Akolkar, R., Auburn, R., Bodell, M., Burnett, D.C., Carter, J., McGlashan, S., Lager, T., Helbing, M., Hosn, R., Raman, T.V., Reifenrath, K., Rosenthal, N., Roxendal, J.: State Chart XML (SCXML): State Machine Notation for Control Abstraction, https://www.w3.org/TR/scxml/.
66. Stănescu, I.A., Stefan, A., Kravcik, M., Lim, T., Bidarra, R.: Interoperability strategies for serious games development. In: Internet Learning. pp. 33–40. DigitalCommons@APUS (2013).
67. IEEE Learning Technology Standards Comittee: IEEE Standard for learning object metadata. IEEE Stand. 1484, 2004–2007 (2002).
68. ADLnet: SCORM, http://www.adlnet.org/scorm/.
69. Henning, P.A., Heberle, F., Fuchs, K., Swertz, C., Schmölz, A., Forstner, A., Zielinski, A.: INTUITEL - Intelligent Tutoring Interface for Technology Enhanced Learning. Int. Work. Pers. Approaches Learn. Environ. 4 pp. (2014).
70. Szentes, D., Bargel, B.-A., Streicher, A., Roller, W.: Enhanced test evaluation for web based adaptive learning paths. 2011 7th Int. Conf. Next Gener. Web Serv. Pract. 352–356 (2011).
71. Minovic, M., Milovanovic, M., Starcevic, D., Minović, M., Milovanović, M.: Using Learning Objects in Games. In: Lytras, M., Ordonez De Pablos, P., Ziderman, A., Roulstone, A., Maurer, H., and Imber, J. (eds.) Knowledge Management, Information Systems, E-Learning, and Sustainability Research SE - 33. pp. 297–305. Springer Berlin Heidelberg (2010).
72. adelbla, Marchiori, E., EUCM-Developer, Martinez, Torrente, J., Moreno-Ger, P.: eAdventure, http://sourceforge.net/projects/e-adventure/, (2015).
73. Silva, J., Teixeira, F., de Jesus, E., Sá, V., Fernandes, C.T.: A taxonomy of educational games compatible with the LOM-IEEE data model. Proc. Interdiscip. Stud. Comput. Sci. SCNIENTIA. 44–59 (2008).
74. El Borji, Y., Khaldi, M.: An IEEE LOM Application Profile to Describe Serious Games «SG-LOM». Int. J. Comput. Appl. 86, 1–8 (2014).
75. Object Management Group: Unified Modeling LanguageTM (UML®) Resource Page, http://www.uml.org/.
76. Rucker, R.: Software Engineering and Computer Games. Addison-Wesley, Harlow, Essex (2003).
77. Gregory, J.: Game Engine Architecture. A K Peters/CRC Press, Boca Raton, FL, USA (2014).
78. Anderson, E.F., Engel, S., Comninos, P., McLoughlin, L.: The case for research in game engine architecture. Proc. 2008 Conf. Futur. Play Res. Play. Share - Futur. Play '08.

228–231 (2008).
79. Göbel, S., Salvatore, L., Konrad, R.: StoryTec: A Digital Storytelling Platform for the Authoring and Experiencing of Interactive and Non-Linear Stories. In: Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS '08. International Conference on. pp. 103–110 (2008).
80. Jumail, Rambli, D.R.A., Sulaiman, S.: G-Flash: An authoring tool for guided digital storytelling. In: Computers Informatics (ISCI), 2011 IEEE Symposium on. pp. 396–401 (2011).
81. Yessad, A., Labat, J.M., Kermorvant, F.: SeGAE: A serious game authoring environment. Proc. - 10th IEEE Int. Conf. Adv. Learn. Technol. ICALT 2010. 538–540 (2010).
82. Huizinga, J.: Homo Ludens. Routledge & Kegan Paul, London, Boston and Henley (1949).
83. Orkin, J.: Three states and a plan: the AI of FEAR. Game Dev. Conf. 2006, 1–18 (2006).
84. Majchrzak, K., Quadflieg, J., Rudolph, G.: Advanced Dynamic Scripting for Fighting Game AI. In: Entertainment Computing – ICEC 2015 14th International Conference, Trondheim, Norway, Proceedings. pp. 86–99. Springer International Publishing, Cham (2015).
85. Yannakakis, G.N.: Game AI Revisited. Proc. 9th Conf. Comput. Front. 285–292 (2012).
86. Rival Theory: RAIN AI for Unity, https://www.assetstore.unity3d.com/en/#!-/content/23569, (2014).
87. Millington, I.: Game Physics Engine Development. Morgan Kaufmann Publishers Amsterdam (2010).
88. Anthony: High-performance physics in Unity 5, http://blogs.unity3d.com/-2014/07/08/high-performance-physics-in-unity-5/.
89. Cowan, B., Kapralos, B.: A Survey of Frameworks and Game Engines for Serious Game Development. In: Advanced Learning Technologies (ICALT), 2014 IEEE 14th International Conference on. pp. 662–664 (2014).
90. Epic Games: Unreal Engine, http://www.unrealengine.com/, (2015).
91. YOYOGames: Gamemaker, http://www.yoyogames.com/gamemaker.
92. Marks, S., Windsor, J., Wünsche, B.: Evaluation of game engines for simulated surgical training. In: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia - GRAPHITE '07. pp. 273–280. ACM, New York (2007).
93. Rocha, R., Araújo, R.: Selecting the Best Open Source 3D Games Engines. Proc. brazilian Symp. Games Digit. Entertain. Florianópolis, St. Catarina. 333–336 (2010).
94. blender, https://www.blender.org/.
95. Crystal Space, http://www.crystalspace3d.org/main/Main_Page.
96. delta3d, http://www.delta3d.org/.
97. Gebhardt, N., Stehno, C., Davidson, G., Celis, A.F., Hoschke, L., MacDonald, C., Zeilfelder, M., Nadrowski, P., Hilali, A., Wadsworth, D., Alten, T., Jam, Goewert, J.: Irrlicht 3D Engine, http://irrlicht.sourceforge.net/.
98. jMonkeyEngine, http://jmonkeyengine.org/.
99. OGRE, http://www.ogre3d.org/, (2001).
100. OpenSceneGraph, http://www.openscenegraph.org/.

101. Walt Disney Imagineering Carnegie Mellon University: Panda3D, http://www.panda3d.org/.
102. Petridis, P., Dunwell, I., Panzoli, D., Arnab, S., Protopsaltis, A., Hendrix, M., Freitas, S.: Game Engines Selection Framework for High-Fidelity Serious Applications. Int. J. Interact. Worlds. 2012, 1–19 (2012).
103. Crytek GmbH: CryEngine, http://cryengine.com/, (2015).
104. Valve: Source Engine, (2014).
105. Westhoven, M., Alexander, T.: Towards a Structured Selection of Game Engines for Virtual Environments. In: Shumaker, R. and Lackey, S. (eds.) Virtual, Augmented and Mixed Reality 7th International Conference, VAMR 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings. pp. 142–152. Springer International Publishing, Cham (2015).
106. Sarhan, A.: The utilisation of games technology for environmental design education, PhD thesis, University of Nottingham. (2012).
107. Chen, M.D.: Game Making Tools Round Up, http://markdangerchen.net/2015/08/27-/game-making-tools-round-up/.
108. Electronic Arts Inc: SimCity, www.simcity.com.
109. Cifaldi, F.: Breaking down SimCity's Glassbox engine, http://www.gamasutra.com/-view/news/164870/gdc_2012_breaking_down_simcitys_.php.
110. Willmott, A.: Inside GlassBox, http://www.andrewwillmott.com/talks/inside-glassbox.
111. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture Volume 1: A System of Patterns. Chichester, West Sussex, England: John Wiley & Sons Ltd (1996).
112. OTOY Inc: OTOY, https://home.otoy.com/.
113. Messaoudi, F., Simon, G., Ksentini, A.: Dissecting games engines: The case of Unity3D. Netw. Syst. Support Games (NetGames), 2015 Int. Work. 1–6 (2015).
114. Cooper, K.M.L., Scacchi, W. eds: Computer Games and Software Engineering. Chapman & Hall/CRC, Boca Raton, FL, USA (2015).
115. Hocking, J.: Unity in Action: Multiplatform Game Development in C# with Unity 5. Manning Publications, Shelter Island, NY (2015).
116. Nystrom, B.: Game Programming Patterns. Genever Benning (2014).
117. Schuller, D.: C# Game Programming: For Serious Game Creation. Cengage Learning PTR, Boston, MA, USA (2010).
118. GameDevNet LLC: gamedev.net, http://gamedev.net.
119. AiGameDev.com KG: AIGameDev.com, http://aigamedev.com.
120. Stack Exchange Inc.: Game Development - Stackexchange, http://gamedev.stackexchange.com/.
121. DevMaster LLC: Engines | DevNaster, http://devmaster.net/devdb/engines.
122. clay games: HTML5 Game Engines - Find Which is Right For You, https://html5gameengine.com/.
123. GAMESNETWORK List at LISTSERV.UTA.FI, https://listserv.uta.fi/archives/-gamesnetwork.html.
124. ICEC -- Mailing List of the IFIP Entertainment Computing Community, http://listserver.tue.nl/mailman/listinfo/icec.
125. Mahardy, M.: GDC 2015: Valve Announces Source 2 Engine,

http://www.ign.com/articles/2015/03/04/gdc-2015-valve-announces-source-2-engine.