Journal of Universal Computer Science, vol. 20, no. 4 (2014), 561-582
submitted: 31/7/13, accepted: 28/3/14, appeared: 1/4/14 © J.UCS

Developing Scenario-based Serious Games for Complex Cognitive Skills Acquisition: Design, Development and Evaluation of the EMERGO Platform

Aad Slootmaker
(Open University of the Netherlands, Heerlen, The Netherlands
aad.slootmaker@ou.nl)

Hub Kurvers
(Open University of the Netherlands, Heerlen, The Netherlands
hub.kurvers@ou.nl)

Hans Hummel
(Open University of the Netherlands, Heerlen, The Netherlands
hans.hummel@ou.nl)

Rob Koper
(Open University of the Netherlands, Heerlen, The Netherlands
rob.koper@ou.nl)

Abstract: Serious games are considered to provide powerful and attractive ways to acquire complex cognitive skills for education and training. But existing platforms for development of game-based e-learning often appear either not to be very user-friendly or too rigid or costly. This article addresses the design, development and evaluation of a generic platform for fast and flexible development and delivery of a wide variety of scenario-based games that enables complex cognitive skills acquisition. We present the requirements for the EMERGO platform and which common components it offers to cater for most of the needed functionalities within scenario-based games. We explain how users in various roles can use the platform to manage, develop, deliver and play a broad variety of scenario-based games. Evaluation data are presented to back up the claim that the platform indeed allows for faster, more user-friendly and less costly development and delivery of scenario-based games. Seven years after the platform has been launched, it until now has proven successful and still continues to evolve. We close off with some conclusions and needs for further development.

Keywords: Adaptive eLearning, eLearning Platforms, Technology Enhanced Learning, Game Based Learning
Categories: L.2.0, L.3.0, L.3.6, L.5.1

1 Introduction

Serious games offer a solution for enabling professional learning at a distance, when the acquisition in actual practice would be impossible or rather hard to realize. Professional education requires students to practice complex cognitive skills in authentic professional settings. These skills involve cognitive processes, e.g. problem solving, reasoning, taking decisions or reflecting in context. This kind of *experiential*

education often is difficult to organize in a practical, e.g., because there are more students than internships available or because the supervision of students would be too time-consuming, risky or insufficient in actual practice.

Existing development frameworks for games often are inadequately tuned toward specific learning needs [Nadolski, 12], and game engines often have been developed for just one specific aspect of a game (e.g., graphical rendering). There are frameworks that integrate a number of these more specific engines, but don't support teachers that well in the process of developing serious games, or have a steep learning curve [de Freitas, 10]. For further take-up in education there was a hard felt need to provide teachers with *a user-friendly author environment*. Besides this, existing frameworks often lack suitable logging of game progress, which impedes research on the actual effects of serious games.

The Open University of the Netherlands, being a provider of distance education, has a longer experience in developing serious games for complex cognitive skills acquisition in various content domains and with different learning purposes. These serious games were developed on client computers and delivered on cd/dvd. Not all operating systems were supported, delivery was demanding (reproduction), and technical or functional bug fixes could not be delivered easily. And there was little reuse of game components. Games were mostly built from scratch. There was a need for a platform that would simplify and broaden delivery. The platform should further foster reuse and exchange between serious games for different content domains by offering *reusable and adaptable components* for game development.

Developing serious games is often a costly business. Most games are developed as 3D environments requiring a vast investment in 3D graphics that cannot be reused easily in other games. However, use of 3D is not always needed, because maximum fidelity of the environment does not necessarily lead to better learning [Herrington, 07]. Furthermore, the development and testing of the didactic scenarios of serious games is quite time consuming, because the intended complex skills require many steps to take and many hours to acquire. There was a need for an approach and platform that would support *more cost-effective* development of scenario-based serious games.

Some ten years ago the need for a user-friendly author environment providing teachers with reusable and adaptable components to develop serious games cost-effective was commonly felt in many higher education institutions. This need then was expressed in the development of a number of online platforms that enabled teachers to develop their own serious games without programming. Examples are Fablusi (http://www.fablusi.com/), Unigame (http://www.unigame.net/) and Cyberdam (http://www.cyberdam.nl/). These platforms enabled the development of multi-role-playing games where learners take on the role profiles of specific characters or representatives of organizations. However, our focus was broader than just role-play. We wanted to offer a rich environment for experiential education where students mostly learn on their own and where other actors are mostly implemented as non-playing characters.

The central *research question* of this article is *how to design and develop a generic platform for fast and flexible development and delivery of a wide variety of scenario-based serious games which enable complex cognitive skills acquisition*. According to [Westera, 01] cognitive skills are skills that involve mental processes

that occur in the mind while using, transforming or supplementing available knowledge. *Complex cognitive skills* are associated with higher-order activities like problem solving, reasoning, thinking, assessing and concluding. They include the mental processes of analysis, synthesis and evaluation to produce a re-ordering or extension of the existing cognitive structure. *Scenario-based serious games* are games where learners are placed in complex problem spaces, which mimic real world situations. They are confronted with ill-defined problems, often allowing multiple solutions and requiring application of necessary methodologies or tools and collaboration with fellow learners [Westera, 08]. To enable the acquisition of these complex cognitive skills and this type of games the scenario describes the problem space, which activities have to be done, which materials are needed and how the problem space should be adjusted while the student is playing.

To answer the research question, the remainder of this article will be structured as follows. In section two we elaborate on the type of scenario-based games the platform supports. In section three we present the requirements for the platform. In section four we describe how we developed the platform and present the history of versions. In section five we present the platform roles, the domain model and common reusable components and their underlying generic design. In section six we evaluate if the platform satisfies the requirements and compare it to related work. In section seven we summarize our findings and present our plans for future work.

2 Scenario-based serious games supported by the platform

The platform supports games where the student works as a trainee in an immersive virtual environment that resembles real-life environments like a law firm or an office environment. His virtual supervisor will give him assignments, and will react to and reflect on his outcomes. He will meet virtual experts or other people to gain background knowledge about the skills to acquire. Within the environment the student has a tablet with apps that provide background materials, enable communicating with virtual persons and other students, and help the student to acquire the skills. The student will be confronted with the consequences of his acts. This means that the environment must be able to respond to student actions by giving clear feedback, and adjust itself according to the progress of the student.

Within a game on Sexology for instance, the student attends two patient interviews and a multidisciplinary meeting, and interviews four subject matter experts. He has to learn to prepare himself for the patient interview, to write a summary of the interview, to work out a model related to the causes of the patient's problem, and to write a proposal for treatment. The student starts the game on a square with buildings related to the Sexology course: a hospital, a university, a school, a health service, an aids center and a station (see Figure 1). The station is used to visit virtual patients at home. Within the hospital the student finds his supervisor, subject matter experts, rooms for patient interviews and meetings, and his own room. He has a notepad to make contextualized notes and a recorder to record parts of interviews. On his tablet the student finds background materials like the patient records, a log containing all notes made with the notepad, an app with all recordings made during interviews, a manual explaining the interface of the game and an email app to get mails and send in assignment outcomes.

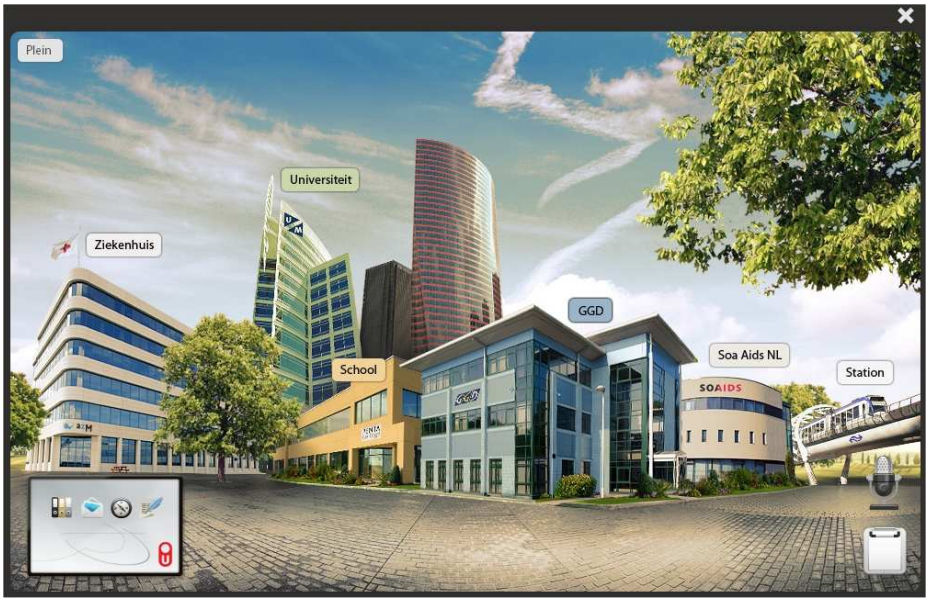

Figure 1: Screen of a game showing a square with buildings to visit. On the bottom left corner we see an icon for the tablet. On the bottom right corner we see a mike to record parts of interviews and a notepad to make contextualised notes

3 Requirements for the platform

The *objective* of the platform is to enable the *fast and flexible development and delivery of a wide variety of scenario-based serious games which enable complex cognitive skills acquisition*. Intended users of the platform are teachers, students, administrators and programmers. *Teachers* will develop games by writing a game scenario, selecting relevant educational material and using the platform to enter game data, game materials and game script, and they will monitor students; *Students* will use the platform to play games; *Administrators* will manage platform users; and *Programmers* will extend the platform. Based on our experience and studies carried out by others [Aldrich, 05], as well as on aforementioned problems with current development, we now list following functional (F) and non-functional (N) requirements for the platform (Table 1).

F1 Offer *teachers* an *intuitive and user-friendly author environment* where they independently can *create and edit games*.

F2 Enable *teachers* to *create and edit game roles*, so students playing together in one game can have different roles.

F3 Offer *teachers* a set of *common reusable and adaptable components* that covers most of the needed functionalities to acquire complex cognitive skills using scenario-based serious games. Teachers should be able to

	select components they need and *edit* these now called *game components*.
F4	Enable *several teachers working together on the same game* so work can be divided.
F5	Enable *teachers* to *preview games* or a single *game component* as a student, at any stage of the development process.
F6	Enable *teachers* to *test games* as a student at any stage of the development process and starting from multiple points within the game script.
F7	Enable *teachers* to *import and export games* so games can be distributed to other platform instances and their content can be reused.
F8	Enable *teachers* to *import and export game components* so game content can be reused.
F9	Enable *teachers* to *monitor progress* of students.
F10	Enable *teachers* to *interfere in a running game*, for instance if outcome quality is insufficient or if a student is stuck in the game.
F11	Offer *students* an *intuitive immersive player environment* where they *play* developed *games*. The player environment should be adjusted according to the actions and progress of a student by using game script.
F12	Enable to *save and persist all student actions*, for game script to operate on, and for evaluation and research purposes.
F13	Enable *students* to *send in* assignment *outcomes*, allowing progression within the game (triggered by game script) and monitoring of progress.
F14	Enable *students* to *enrich* the *running game* with user generated content and share this content with other students.
F15	Enable *administrators* to *manage platform users* and their roles.
F16	Enable *administrators* to *manage game runs*, by assigning a cohort of students to a run and assigning students to game roles.
F17	Enable *administrators* to *manage game teams*, teams of students operating within the same game run.
F18	Enable *programmers* to easily *extend* the platform *with* new *languages*.
F19	Enable *programmers* to rather easily *extend* the set of common reusable components *with* new *components*.
F20	Enable *programmers* to *extend* the player environment *with* new *skins,* to be able to offer (external) parties their own look and feel.
N1	Be *reliable and stable*.
N2	Be *usable on multiple operating systems,* e.g., at and across institutions.
N3	Offer *efficient development and delivery of games*. Delivering and updating the platform and developed serious games should be easy and not affect student's progress.
N4	Be *backward compatible*, authoring and playing of earlier developed games should be possible.
N5	Be *integrated with institutional infrastructures*.

Table 1: Functional (F) and non-functional (N) requirements for the platform

Requirements F3 and F11 directly relate to acquiring complex cognitive skills. Learners will perform authentic tasks in an environment that challenges and makes them curious, presents appropriate and unambiguous outcome goals and provides

clear, constructive and encouraging feedback [Nadolski, 12]. Requirements F1, F5 and F6 relate to aforementioned need for a more user-friendly author environment. Requirements F3, F7 and F8 relate to the need for reusable and adaptable components. Requirements F1, F2 till F8, and N3 relate to the need for more cost-effective development. The requirements are elaborated in a use case diagram (see Figure 2).

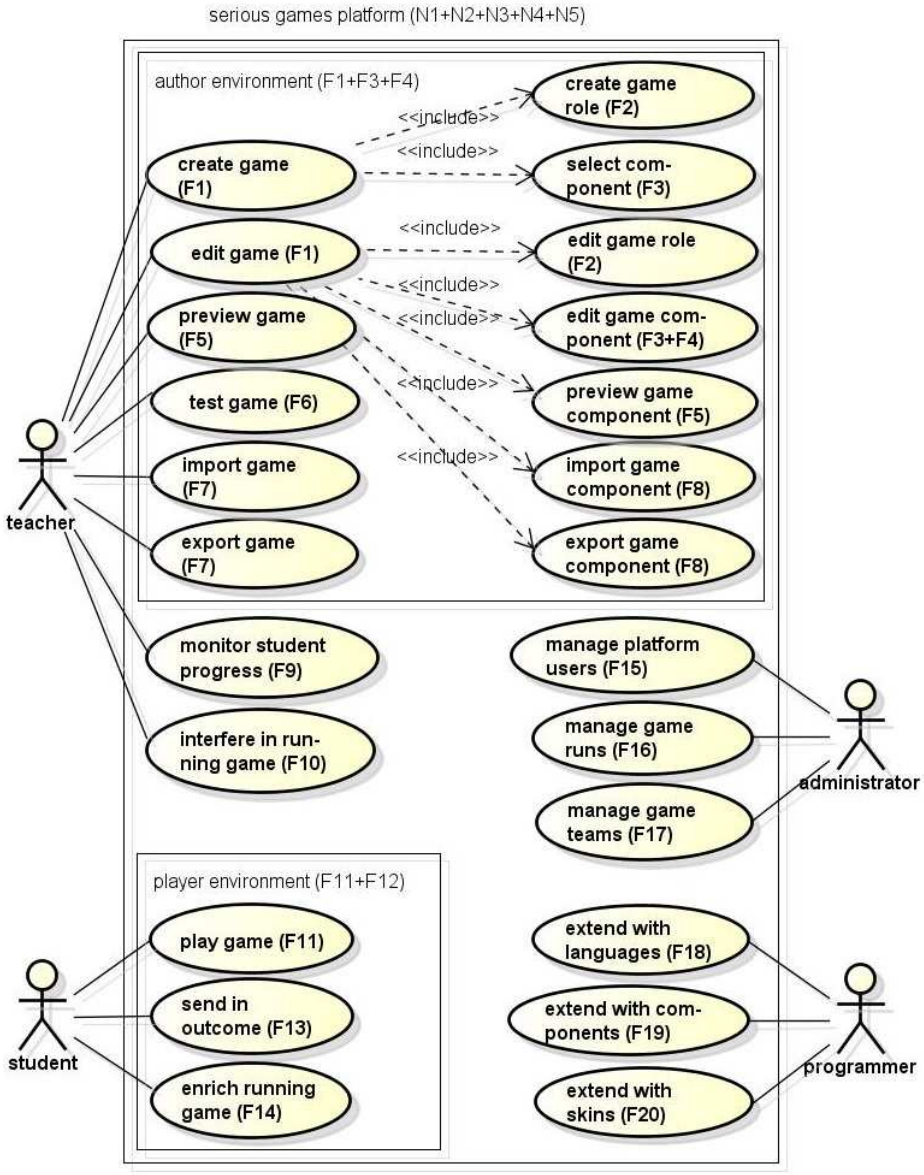

Figure 2: Use case diagram for the platform. Requirements are indicated

Rectangles indicate the boundaries of the author and player environment. These boundaries are debatable. For instance previewing and testing a game could be done outside of the author environment, but we feel these options should be an integral part of it. In the next section we elaborate on the development of the platform.

4 Development of the platform

Version 1 of the platform was developed within the EMERGO project (2006-2007) that was co-funded by SURF foundation, and was intended to be used by all SURF members. The project had three outcomes: a methodology to support writing the scenario for scenario-based serious games [Nadolski, 08], a platform for developing and delivering the games and five games that were used in education. This article will focus on the *EMERGO platform*. *Version 2* of the platform was one of the outcomes of the Skills Labs project, also co-funded by SURF foundation, and was released in 2010. The project also delivered four games that were used in education. *Version 3* was an outcome of a couple of projects and was released in 2013. The platform is Open Source and can be found on SourceForge [EMERGO, 13].

The first development step was to choose an application architecture. We choose for a *multi-tier client-server architecture*, because one tier can be substituted by another implementation without affecting the other tiers. To meet requirement F12 (save and persist all student actions) we choose to use a *centralized database* on a server so game script, also located on a server, can operate on student actions, and student data can be shared within multi-role games and is easily available for evaluation and research. To meet requirements N1 (reliable and stable) and N2 (usable on multiple operating systems), and because we had broad experience with it, we choose the *Java EE platform*. To meet requirement N3 (efficient development and delivery of games), we choose the client to be *web-based*, requiring no installation of dedicated client software to develop or play a game and enabling easily updating the platform and developed games. To meet requirement N1 (reliable and stable) we choose the *Spring application framework* [Spring framework, 13], to implement our domain model and business logic, and the MySQL database server for data persistence in a centralized database. Both are proven technology and widely used within the Open Source community. For the client web interface we choose *ZK framework* [ZK framework, 13] that runs on all common browsers. ZK framework is a so called RIA (Rich Internet Application) offering the same interactivity and responsiveness as a desktop application, and therefore offered the best guarantee to meet requirements F1 (intuitive and user-friendly author environment) and F11 (intuitive immersive player environment). ZK comes with a very rich set of visual components, which offered the best guarantee to be able to build our own components, meeting requirements F3 (common reusable and adaptable components) and F19 (extend with new components). ZK is very fast and Ajax based, so all student actions can be saved immediately, meeting requirement F12 (save and persist all student actions).

The platform was *developed by a multidisciplinary team* of educational technologists, interaction designers and programmers. For the development process we used an *agile methodology* similar to Scrum, implying always delivering working

software, short iterations, quick response to change and close cooperation within the development team.

We started the development process with the *design of the platform*, which involved following five steps: (1) Identify needed *platform roles*; (2) Create a *domain model* for the platform; (3) Identify needed *common reusable components*, meeting requirement F3 (common reusable and adaptable components); (4) Create a *generic component design*, meeting requirement F19 (rather easily extend with new components); and (5) Design the *component for handling game script*, meeting requirement F11 (using game script, the player environment should be adjusted). In the next section we will present the results of these five design steps.

Next we started the *implementation of the platform*. After implementing the domain model and business logic we could start *implementing the use cases in a certain order*. Most use cases depend on each other, e.g., before you can create a game, you must first be added as a platform user. While implementing the use cases, we also started *implementing components in a certain order*, determined by their mutual dependency and by the priority within the development team. Version 1 of the platform contained an initial set of common components. This set was extended with new components in version 2 and version 3.

The *evaluation of the platform* involved measuring if requirements F1 (intuitive and user-friendly author environment), F11 (intuitive immersive player environment) and N3 (efficient development of games) were satisfied. The evaluations of the other requirements were based on our experiences with the users of the platform, ourselves included. Versions 1 and 2 of the platform were evaluated on the aspects of intuitivity and user-friendliness for teachers using the author environment to enter data. Both versions were evaluated on the production ratio for developed games and on student satisfaction with the user-interface of the player environment. Besides this, version 1 was evaluated on student satisfaction, and version 2 on the aspects of quality, studiability and effectiveness of developed games as perceived by students. Intuitivity and user-friendliness as perceived by teachers were operationalized by 'the capacity to use the platform components independent without help' and 'the simplicity encountered when using platform components to enter data', respectively. Intuitivity and user-friendliness were measured using a questionnaire containing questions, like 'Were you able to use the component independently?' and 'How simple was it to use the component?'. Production ratio (as main indicator for efficient development) was measured by comparing development hours (as were recorded in the project administration) with the estimated or measured study time. Student satisfaction was operationalized and questioned as the appreciation of the player environment. Quality and studiability were operationalized in twenty two questions, like 'Were the instructions for performing a task clear enough?' and 'Did you get enough background material to perform a task?'. Effectiveness of developed games was determined by students' grades, in one case also by comparing them with grades obtained in classroom education.

5 Design of the platform

In this section we present the design of the platform; the platform roles, the domain model, the implemented common reusable components, the underlying generic component design and the script component.

5.1 Platform roles

Starting from the use case diagram defined in section three (Figure 2) we identified *five platform roles* that should have their own working environment within the platform: *administrator, developer, run manager, tutor and student*. The administrator and run manager platform role are best filled in by user 'administrator'. The developer and tutor platform role are filled in by the user 'teacher'. The student platform role is filled in by the user 'student', or if a teacher has a role within the game, by the user 'teacher'. The user 'programmer' has no counterpart as platform role, he has his own development environment to extend the platform.

The *administrator* platform role *manages* all *users* and their platform *roles* (requirement F15). Further he can help students who get technically stuck in a game, by *inspecting* a student's *progress* in the player environment, and *adjusting* his *progress* if necessary (requirement F10). If for instance certain materials don't become available for a student, due to a bug, the administrator can make them available.

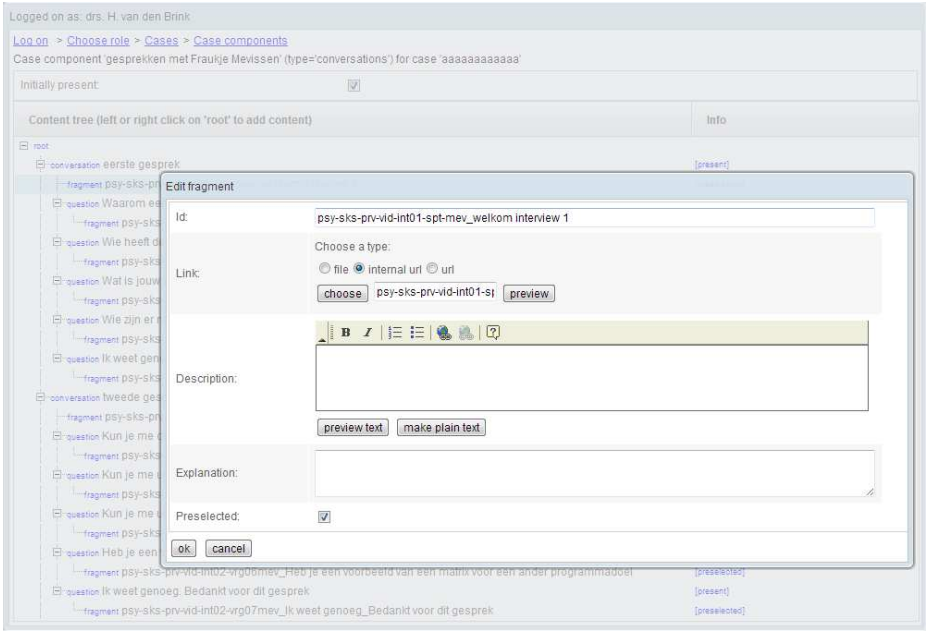

Figure 3: Game component content editor showing a dialogue screen to enter a conversation fragment

The *developer* uses the *author environment* to create and edit *games* (requirement F1). Per game he can create and edit *game roles* (requirement F2) and *game components* by selecting components to use and enter their content (requirement F3). If needed the *game owner* (the developer who created the game) can *assign* other developers as *author* of certain game components (requirement F4). All game component content is entered using one editor (see Figure 3). During authoring the developer can preview the game or a game component in the player environment (requirement F5). And he can test the game in the player environment from multiple points within the game script (so in time) (requirement F6) and for every game role, and even can test with multiple players. Finally he can import and export a game or a game component as an IMS content package [IMS, 07] (requirements F7 and F8).

The *run manager* creates and updates *runs* of developed games (requirement F16) and he defines *run users* by assigning users to a run. Further he can run users to a certain game role and define *run teams* of run users if appropriate (requirement F17).

The *tutor* monitors the progress of students (requirement F9). He gets overviews of tasks students have completed and assignment outcomes they have submitted. If needed, he can interfere in the game by sending an email as if it is sent by a non-playing character (requirement F10), so students don't notice the difference. This way thresholds can be raised, e.g., to guarantee the quality of students outcomes. Further he can help students who get stuck in a game by *inspecting* a student's *progress* in the player environment and instructing how to proceed (requirement F10).

The *student* sees an overview of games to play and can start the *player environment* (see Figure 1) with a chosen game (requirement F11). The player environment renders all developed games in 2D, and mimics the professional practice students later have to work in. All student *progress* is saved and persisted continuously (requirement F12).

5.2 Domain model

The resulting domain model (see Figure 4) shows all entities of the platform and how they are related. *Components* are the most important concept of the platform. Components are used to build and play a game. Programmers maintain the set of components and can extend it. *Users* of the EMERGO platform can get multiple platform roles. As an *administrator*, a User can manage Users and give them platform roles. As a *developer*, a User can manage multiple *Games* and is the owner of the Games he creates. Per Game he is the author of multiple *Game Roles* and *Game Components*. He can make other developers author of his Game Components. The Game itself is not much more than a container for Game Roles and Game Components. Components can have multiple Game Component instances and a certain Game Component can be used by multiple Game Roles. As a *run manager*, a User manages *Runs*. A Game can have multiple Runs. The run manager allocates Users to a Run as *Run Users*. He also can create *Run Teams* of Run Users. As a *tutor*, a User can monitor Runs. As a *student*, a User can participate in multiple Runs as Run User and can be member of multiple Run Teams. A Run User has *Run User Progress* within a Run and a Run Team has *Run Team Progress*. Note that both type of progress can be present in one Run. Progress is related to a Game Component.

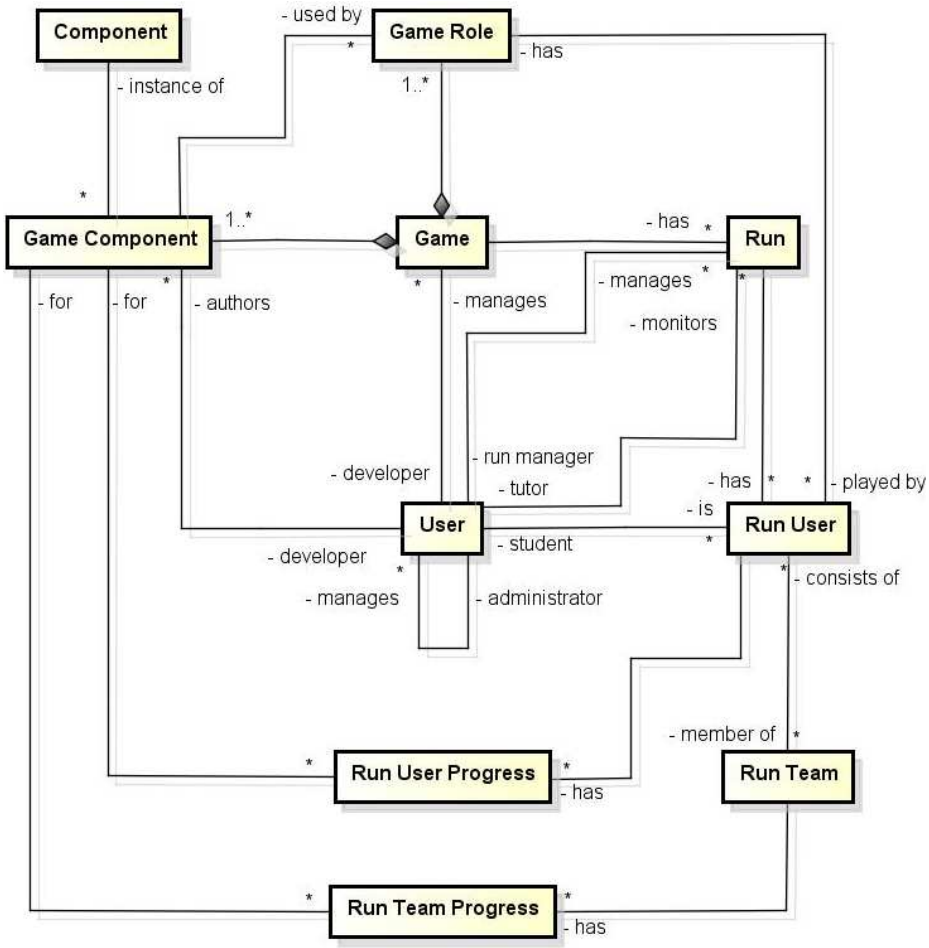

Figure 4: Domain model of the platform

5.3 Common platform components

Based on our experience in developing scenario-based serious games over the years, we have identified a number of *components* that represent *common functionalities* for this kind of games. Students are always placed in an environment with multiple locations where they can interview people, and have a virtual tablet with apps to help them with their assignments. Table 2 lists all components that we have implemented and in which version of the platform.

Component	**Function**	**Version**
Locations	Navigate through the game and stage setting	1
Navigation	More naturally navigate through the game, using hyper regions on location backgrounds and the parallax effect (see Figure 1)	3
Conversations	Interact with non-playing characters on location, using video	1
Alerts	Provide popup instructions	1
Notepad	Make contextualized notes. Available on every location	1
Memo recorder	Record parts of interviews. Available on every location	3
Profile	See each other's profile and scores defined in the 'Scores' component. Available on every location	3
Chat	Chat in game. Available on every location	3
Tablet	Provide available apps. Available on every location	1
Assessments	Enable in game assessment, using items defined in 'Items' component. App on tablet	1
Directing	Examine an interview using different camera angles. App on tablet	3
Email	Enable in game email, e.g., for providing predefined assignments to students and sending in assignment outcomes by students. App on tablet	1
Google Maps	Enable showing maps with markers. App on tablet	2
Logbook	Provide overview of notes made with the 'Notepad' component. App on tablet	2
Memo player	Look back interview recordings. App on tablet	3
Resources	Provide background material. App on tablet	1
Tasks	Provide tasks overview or to do list. App on tablet	1
Video manual	Explain the player environment interface. App on tablet	3
Items	Provide item bank of multiple choice and multiple answer questions to be used in the 'Assessments' component	1
States	Enable defining game properties that can be read and changed in game script	3
Scores	Enable defining scores to be shown in the 'Profile' component	3
Script	Enable dynamical adjustment of the player environment using game script	1
Relations	Store relations between content of different components	1

Table 2: Common components, their function and in which version of the platform they were implemented

The last two components don't represent game functionalities, but are added because they are common in every game. The *Script component* is used by developers to enter the game script. The *Relations component* is used by the platform to store relations between content of different game components, e.g., which items belong to a certain assessment.

5.4 Generic component design

To be able to meet requirement F19 (rather easily extend with new components), we wanted the domain model to remain unchanged if we extend the platform with a component. We therefore choose to store all component related content in XML. It concerns the component itself, the game component content entered by developers and the game component progress of students, as can be seen in the domain model.

To be able to meet Requirement F19 (rather easily extend with new components), we had to come up with a *generic design for components*, so components could be added in the future too. We choose to *define every component by an XML definition* (see example in Figure 5), that includes:

1. component *properties* (e.g., a component is present for a student or not);
2. *relations* with other components (e.g., the Logbook component will show all notes entered in the Notepad component);
3. possible *content elements*, that make up the content of a component (e.g., locations, folders, resources, interviews, questions);
4. mutual *hierarchy* of content elements, indicating which content element must be part of another one (e.g., questions are part of an interview);
5. *relations with other content elements* (e.g., an item belongs to an assessment);
6. *content to be entered by a developer* (e.g., the text of a question to be asked or a reference to a video stream to be played);
7. *content to be entered by a student*, e.g. (an email text or attachments)
8. content elements' *properties* (e.g., an email is sent);
9. the *type of the properties;*
10. the *default values of the properties*;
11. which *property values can initially be changed by developers*; and
12. which *property values can be read and/or changed by game script.*

Properties have different *purposes*. There are properties that determine *visibility or accessibility* in the player environment (requirement F11). These properties typically can change during the game and are set by developers, initially or by using game script. Other properties determine the *adaptability* of a component (requirement F3) in either functionality or layout, and are initially set by developers. Most properties are used to handle *progress* within the game and are set triggered by student actions (e.g., opening a resource), game script (e.g., sending a predefined email) or the platform itself (keeping game time). We have defined over thirty properties. Table 3 lists properties that are used most often.

```
<data>
 <component type="root">
  <status/>
  <initialstatus attributes=""/>
  <getstatus attributes=""/>
  <setstatus attributes=""/>
 </component>
 <content type="root" childnodes="alert" maxid="">
  <alert id="" type="node" childnodes="" key="pid" multiple="true">
   <pid type="line" private="true"/>
   <richtext type="simplerichtext"/>
   <explanation type="text" private="true"/>
   <status present="true" selected="false" opened="false" sent="false"/>
   <initialstatus attributes=""/>
   <getstatus attributes="present,selected,opened,sent"/>
   <setstatus attributes="present,sent"/>
  </alert>
 </content>
</data>
```

Figure 5: Simple example of an XML definition: the Alerts component

Property	**Type**	**Purpose**	**Example**
Present	Boolean	Does a student see a component or content element?	A tablet app is present or not
Accessible	Boolean	Can a student access a component or content element?	A door is locked or not
Expandable	Boolean	Can a student expand a content element?	A resource folder can be expanded or not
Expanded	Boolean	Is a content element expanded by a student?	A resource folder is expanded or not
Opened	Boolean	Is a component or content element opened by a student?	A door is opened
Started	Boolean	Is a component or content element started by a student or the platform?	A video stream is started by the platform
Finished	Boolean	Is a component or content element finished by a student or the platform?	An assessment is finished by a student
Sent	Boolean	Is a content element sent by a student or the platform?	An email is sent by a student or the platform

Table 3: Most used properties and their purpose

All game component content entered by game developers and all game component progress of students is stored in XML, in a structure defined by the XML definition of

the corresponding component. Progress is formed by all property changes in time and possibly associated content like an email text and attachments entered by a student. Some components allow a developer to set properties that enable students to create and share user generated content (requirement F14). This content is saved within progress too.

The generic component design assures that adding new components has a minimal effect on the author environment. Only if a new component demands a new content format, a corresponding input element has to be added in the game component content editor. This however does not account for the player environment. It has to be extended with an embedded player for the component.

5.5 The Script component

By using the Script component a developer enters the dynamics of the game scenario, thus determining how the player environment should be adjusted according to the actions and progress of a student. Conditions and actions are entered using dialogues that require no programming (requirement F1). A condition and its related actions resemble an 'if-then' statement in a programming language (see Figure 6).

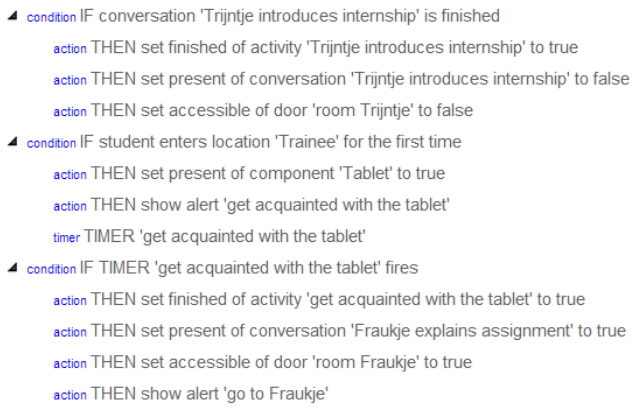

Figure 6: An example of script (entered for the game described in section two). Conditions and actions are added using dialogue screens

A script *condition* enables the developer to check whether properties have been set to certain values, e.g. if a student has opened a location then its opened property is set true. A condition can be built up by sub conditions using logical operators. Conditions are triggered by events, either by student actions or timer events, resulting in a property change. If the condition becomes true its related actions will be executed.

A script *action* enables the developer to set a property to a certain value, e.g., a new conversation can be made available by setting its present property to true. When a property is set, the execution of a script action can result in other conditions being triggered. A special kind of script action is the definition of a script *timer*. If its 'parent' condition becomes true, the timer will start. Another condition then can be used to check if the timer fires. Timers have a certain delay, can be defined to be repetitive, and can measure game-time or real-time.

Conditions and actions themselves have properties too. One of them is the present property. By setting its value to true or false a developer can switch conditions and actions on and off, meaning the working of the script itself can be changed too. The Script component only allows conditions and actions to be defined on existing content entered by developers, not on user generated content entered by students.

In the next section we present the evaluation of the platform and its relation to other work.

6 Evaluation of the platform and related work

6.1 Evaluation of the platform

The EMERGO platform has been used in various projects with both internal and external partners. In seven years, twenty two games were developed, which were used in education by nearly 4000 students in total. Games were developed for six content domains, had a broad variety in scenarios and structure and differed both in complexity and study load, ranging from 2 to 30 hours. Twenty games were single user games and two games were multi-role games that involved collaboration between students. The platform currently is being used by five educational institutions.

We evaluated requirements F1 (intuitive and user-friendly author environment), F11 (intuitive immersive player environment) and N3 (efficient development of games) for nine developed games, five running on version 1 of the platform and four on version 2. All nine games were of the same type as described in section 2. The teachers developing with version 1 were different from the ones developing with version 2. Teachers originated from two educational institutions and had a background in Environmental Sciences. Nadolski et al. [Nadolski, 08] evaluated version 1 of the platform and found that teachers only had trouble using the Script component independently (one out of three) and that the Script and Conversations components were most difficult to use. They also found that students (n = 8) were very satisfied with the user interface of the platform and with the developed games. Furthermore they found an average production ratio of 1:25 (1 hour study load costs 25 hours development time) four five developed games, compared with average production rates of 1:100 and higher found before [Alessi, 01]. Version 2 of the platform was evaluated in the Skills Labs project (for evaluation results see http://dspace.ou.nl/handle/1820/2385). Again teachers only had trouble using the Script component independently (one out of four), and found the Script and Conversations components most difficult to use. Students (n = 40) were satisfied with the user interface of the platform. The average production ratio for four developed games was 1:30. Version 2 was also evaluated regarding quality and studiability, and effectiveness of developed games. Students (n = 40) judged quality and studiability of the four developed games as sufficient (three games) or good (one game). Effectiveness was determined by student's grades. The average grade was sufficient to good, only two students out of forty scored insufficient. For one game grades were compared with grades obtained in classroom education, and were slightly better. Evaluation of the other requirements is based on our own experiences with the users of the platform, ourselves included.

Below we discuss if the functional and non-functional requirements were satisfied.

- F1 (intuitive and user-friendly author environment) was *partly* satisfied. All teachers could author all components independently, except for the Script component. The Script and Conversations component were quite difficult to use.
- F2 (multiple game roles) was satisfied, but only used in two games.
- F3 (common reusable and adaptable components) was satisfied. One component can be used in multiple games and game components can be imported and exported. The generic component design assures that components can be defined to be adaptable.
- F4 (several teachers working together on the same game) was satisfied.
- F5 (preview games and game components) was satisfied. It was an indispensable option while developing games and new platform components.
- F6 (test games) was satisfied. It was an indispensable option for fast development of games and new platform components.
- F7 (import and export games) was satisfied. It turned out to be very handy for distribution of games to other platform instances.
- F8 (import and export game components) was satisfied.
- F9 (monitor progress) was satisfied.
- F10 (interfere in a running game) was satisfied. In some games this option was predesigned in the game scenario. However, the option was mostly used by administrators to help students who were stuck in a game.
- F11 (intuitive immersive player environment) was satisfied. Students were satisfied or very satisfied with the player environment.
- F12 (save and persist all student actions) was satisfied. Students almost never lost data and could always continue a game the next session. A first scientific article based on the logging data is in preparation [Westera, 14].
- F13 (send in outcomes) was satisfied. Outcomes are sent in as an attachment of an in-game email.
- F14 (enrich running game with user generated content) was satisfied. It was implemented for the Resources and Google Maps components.
- F15 (manage platform users) was satisfied.
- F16 (manage game runs) was satisfied.
- F17 (manage game teams) was satisfied.
- F18 (extend with languages) was satisfied. Currently supported languages are English, Dutch and Spanish.
- F19 (rather easily extend with new components) was satisfied. In version 2 and 3, the platform was extended with new common components. The generic component design assures no or very little adjustment of the author environment, although adjustment of the player environment still is time consuming.
- F20 (extend with skins) was satisfied. In version 3, the platform was expanded with the ability to support multiple skins. The current platform has three skins and new skins can be added rather easily.
- N1 (reliable and stable) was satisfied. It is demonstrated by the many games developed and many students playing them.
- N2 (usable on multiple operating systems) was satisfied. The platform currently runs on Windows and Linux servers.

- N3 (efficient development and delivery of games) was satisfied by our choice for a web client, and the abilities to update developed games in case of bugs and to help students who are stuck. Production ratios are better than before.
- N4 (backward compatible) was satisfied. Games developed seven years ago still run on the platform.
- N5 (be integrated with institutional infrastructures) was satisfied. The platform was integrated with the ELO of the Open University to enable single sign-on.

6.2 Related work

During the last decade there were a lot of initiatives to get serious game development on a higher level, strongly supported by the European Commission.

The *ELEKTRA project* (2006 - 2008) for instance, was a research project that focused on bridging the gap between computer science and pedagogy. The project delivered a 3D game on physics meant to engage youngsters for the subject. In-game feedback of these youngsters was used to fine tune the game. The game is analogue to the EMERGO platform in being able to adapt the player environment according to player progress, but differs on being an offline 3D game and not an online development and delivery platform of multiple games.

The *80days project* (2008 – 2010, http://www.eightydays.eu/) was a follow-up of the ELEKTRA project and focused on game adaptation to individual learners, their prior knowledge, abilities, preferences, needs and aims (adaptive personalized learning). On a micro level by giving feedback or hinting in specific learning situations, and on a macro level by sequencing and pacing of learning situations tailored to the individual learner. The project delivered a 3D game on geography which was developed using the StoryTec framework [Göbel, 08], an authoring tool for the development of story-based, process-oriented, interactive 3D applications. It resembles EMERGO in enabling authors to develop games without or with minor programming skills. The Story Editor within StoryTec has some resemblance with the Script component of EMERGO in being able to enter conditional transitions within the game, to go from one scene to another, and to enter actions on content elements. And both platforms enable adaptive personalized learning. But while StoryTec focuses on highly graphical oriented 2D/3D games to be developed and played on a client computer, EMERGO focuses on lesser graphics, use of video and web-based development and delivery. This different focus is related to different customer demands for both platforms.

The *ImREAL project* (2010 – 2013, http://www.imreal-project.eu/), was a European research project focusing on the development of a suite of learning services which extract their data from the real world and can be plugged into virtual environments to augment these environments and enhance self-regulated learning. The learning services were developed by the participating universities. Two existing commercial products were extended to make use of these services. In a first use case an existing role-play simulation environment, developed by EmpowerTheUser (http://www.etu.ie/), was extended to use services related to cultural variations in interpersonal communication, to user generated content, to user profiles (extracted from user activity on the Social Web) and to supporting learners in understanding and improving how they learn. In a second use case another role-play simulation environment, developed by Imaginary (http://www.i-maginary.it/en/), was extended

with a story boarding environment for collecting and structuring content for simulations, and same services as in the first use case. Both commercial simulation environments require no programming, like is the case with EMERGO, and offer rich immersive user experiences, but are not freely available. They support web-based delivery, although it is unclear if all student actions are persisted, but they don't support web-based development. It would certainly be interesting to explore if EMERGO could be extended with the ImREAL learning services.

Another related initiative is the *eAdventure project* (http://e-adventure.e-ucm.es/), a research project of Universidad Complutense de Madrid that delivered the eAdventure authoring tool for the creation of point-and-click adventure games for educational purposes. Developed games can be exported as SCORM package and therefore can be integrated with Learning Management Systems, enabling exchange of adaptation and assessment data. In this respect it is more mature than EMERGO. eAdventure is more focussed on decision making and influencing or adapting certain behaviour, while EMERGO focuses on acquiring complex cognitive skills. Games can be developed on multiple platforms and can be deployed on these platforms and on the web too, although then not all student actions are persisted. It has an easy-to-use game editor, which requires no programming, just like EMERGO, but it does not support multi-role or multi-user games or sharing of content between students.

7 Conclusions and future work

7.1 Conclusions

We demonstrated how to design and develop a generic platform that enables fast and flexible development and delivery of a wide variety of scenario-based serious games which enable complex cognitive skills acquisition.

The platform is *generic* in the sense that it enables a broad variety of game scenarios to be authored, to be played and to be monitored. It offers a set of common reusable components a teacher can pick from to develop a game. The components and their content can be reused in other games. One player environment delivers the variety of scenarios to students and saves and persist all student actions continuously, fostering TEL research on all games.

The platform is *fast* in the sense that teachers can use it mostly independent, can draw on already developed components and can preview and test a game during development and from any point in the scenario, which results in more cost-effective development, as indicated by better production ratios than before. Web-based delivery assures fast and easy delivery of games, updates of games and the platform itself.

The platform is *flexible* in the sense that a game can have multiple authors, a teacher can adjust already deployed games in case of bugs and can interfere in a running game, and the platform provides tooling to help students who are stuck. The platform can be extended rather easily with new components and languages, and skins for the player environment. Developed games can be easily distributed to other platform instances.

Nineteen out of twenty functional requirements were fully satisfied. Requirement F1 (intuitive and user-friendly author environment) was partly satisfied. Entering game script turned out to be too difficult. We could improve its interface, but

scripting still requires more technical skills so probably better could be entered by a programmer. Another way to improve could be using predefined templates or game patterns e.g. collaboration scripts (see next subsection). Although requirement F19 (rather easily extend with new components) was satisfied, we expect that extending the player environment can be improved by constructing it using interface building blocks based on macros or templates. All non-functional requirements were satisfied.

7.2 Future work

Collaboration scripts have been scarcely implemented in serious games so far. Therefore we have built and evaluated two games using online collaboration [Hummel, 11; Hummel, 13]. We will use this experience to extend the EMERGO platform with components that support collaboration. This will involve adding new components for rating, voting and negotiation, and extending the script component to enter and handle collaboration script. We also consider integrating an online conferencing system as an alternative for chat.

We would like to extend the platform with real-time elements (known as augmented virtuality) like web services for presenting real-time data, real-time video with non-playing characters met in video, and sensor data for better support. With regard to the latter option, at the Open University research is done and software is developed for real time emotion recognition using visual and auditory sensors [Bahreini, 12]. To enable research on the learning benefits of real time emotion recognition in serious games, we will integrate this software with the EMERGO platform, so the player environment can be adjusted according to the student's emotions.

We are involved in some projects where the EMERGO platform will be used in developing countries e.g. Kenia, Colombia. In these countries connectivity is a problem, so we will make the platform better suitable for low bandwidths. The platform will buffer game content when sufficient bandwidth is available, to account for low connectivity later on. We consider developing a mobile client app for the player environment in case of no connectivity at all. We then could extend the platform to make use of the capabilities of mobile devices like GPS positioning, and making pictures, video and audio.

We already experimented with integrating the Unity Web Player and the EMERGO platform, by playing a Unity game embedded in the platform and exchanging data between player and platform. The platform then could support students playing an existing Unity game. We would like to further explore this promising possibility.

Acknowledgements

We wish to thank SURF foundation for co-funding the development and scaling-up of the EMERGO platform. We also thank all developers, teachers and students of the institutions contributing to the initial development and extension of the platform.

References

[Aldrich, 05] Aldrich, C. (2005) Learning by doing: the essential guide to simulations, computer games, and pedagogy e-learning and other educational experiences. San Francisco, CA: John Wiley & Sons

[Alessi, 01] Alessi, S. M., & Trollip, S. R. (2001). Multimedia for learning: Methods and development. Needham, MA: Allyn & Bacon.

[Bahreini, 12] Bahreini, Kiavash, Nadolski, Rob, Qi, Wen, Westera, Wim (2012) FILTWAM - a Framework for Online Game-Based Communication Skills Training - Using Webcams and Microphones for Enhancing Learner Support. Proceedings of The 6th European Conference on Games Based Learning, 39-47

[de Freitas, 10] de Freitas, S., Rebolledo-Mendez, G., Liarokapis, F., Magoulas, G., & Poulovassilis, A. (2010). Learning as immersive experiences: Using the four-dimensional framework for designing and evaluating immersive learning experiences in a virtual world. British Journal of Educational Technology, 41(1), 69–85.

[EMERGO, 13] EMERGO project page. (2013) http://sourceforge.net/projects/emergo/

[Göbel, 08] Göbel, S., Salvatore, L., Konrad, R., Mehm, F. (2008). A Digital Storytelling Platform for the Authoring and Experiencing of Interactive and Non-linear Stories. In Spierling, U., Cavazza, M., Peinado, F., Aylett, R., Swartjes, I., Kudenko, D., Young, R., Tychsen, A., Pizzi, D., El-Nasr, M. (eds.) Interactive Storytelling 2008. LNCS, vol. 5334, pp. 325-328. Springer, Berlin / Heidelberg (2008)

[Herrington, 07] Herrington, J., Reeves, T.C., and Oliver, R. (2007). Immersive learning technologies: Realism and online authentic learning. Journal of Computing in Higher Education. 19 (1), 65-84.

[Hummel, 11] Hummel, H. G. K., Van Houcke, J., Nadolski, R. J., Van der Hiele, T., Kurvers, H., & Löhr, A. (2011). Scripted collaboration in serious gaming for complex learning: Effects of multiple perspectives when acquiring water management skills. British Journal of Educational Technology, 42(6), 1029–1041.

[Hummel, 13] Hummel, H., Geerts, W., Slootmaker, A., Kuipers, D., & Westera, W. (2013). Collaboration scripts for mastership skills: online game about classroom dilemmas in teacher education. Interactive Learning Environments, 1–13.

[IMS, 07] IMS Content Packaging Information Model. Version 1.2 Final Specification. Retrieved March 01, 2007, from http://www.imsglobal.org/content/packaging/cpv1p2pd2/imscp_infov1p2pd2.html

[Nadolski, 08] Nadolski, R.J., Hummel, H.G.K., Van den Brink, H.J., Hoefakker, R., Slootmaker, A., Kurvers, H., Storm, J. (2008). EMERGO: methodology and toolkit for efficient development of serious games in higher education. Simulation & Gaming 39(3), 338-352.

[Nadolski, 12] Nadolski, R. J., Hummel, H. G. K., Slootmaker, A., & Van der Vegt, W. (2012). Architectures for Developing Multiuser, Immersive Learning Scenarios. Simulation & Gaming, 43(6), 825–852.

[Spring framework, 13] Spring framework. (2013) http://www.springsource.org/

[Westera, 01] Westera, W. (2001). Competences in education: a confusion of tongues. Journal of Curriculum Studies, 33(1), 75–88.

[Westera, 08] Westera, W., Nadolski, R.J., Hummel, H.G.K., & Wopereis, I. (2008). Serious Games for Higher Education: a Framework for Reducing Design Complexity. Journal of Computer-Assisted Learning, 24(5), 420–432.

[Westera, 14] Westera, W., Nadolski, R.J., Hummel, H.G.K. Serious Gaming Analytics: What Students´ Log Files Tell Us about Gaming and Learning, in preparation, 2014.

[ZK framework, 13] ZK framework (2013) http://www.zkoss.org/