Software Components for Serious Game Development

Wim Westera[1], Wim van der Vegt[1], Kiavash Bahreini[1], Mihai Dascalu[2], and Giel van Lankveld[1]
[1]Open University of the Netherlands, Heerlen, The Netherlands
[2] University Politehnica of Bucharest, Bucharest, Romania
wim.westera@ ou.nl
wim.vandervegt@ou.nl
kiavash.bahreini@ou.nl
mihai.dascalu@cs.pub.ro
giel.vanlankveld@ou.nl

Abstract: The large upfront investments required for game development pose a severe barrier for the wider uptake of serious games in education and training. Also, there is a lack of well-established methods and tools that support game developers at preserving and enhancing the games' pedagogical effectiveness. The RAGE project, which is a Horizon 2020 funded research project on serious games, addresses these issues by making available reusable software components that aim to support the pedagogical qualities of serious games. In order to easily deploy and integrate these game components in a multitude of game engines, platforms and programming languages, RAGE has developed and validated a hybrid component-based software architecture that preserves component portability and interoperability. While a first set of software components is being developed, this paper presents selected examples to explain the overall system's concept and its practical benefits. First, the Emotion Detection component uses the learners' webcams for capturing their emotional states from facial expressions. Second, the Performance Statistics component is an add-on for learning analytics data processing, which allows instructors to track and inspect learners' progress without bothering about the required statistics computations. Third, a set of language processing components accommodate the analysis of textual inputs of learners, facilitating comprehension assessment and prediction. Fourth, the Shared Data Storage component provides a technical solution for data storage - e.g. for player data or game world data - across multiple software components. The presented components are exemplary for the anticipated RAGE library, which will include up to forty reusable software components for serious gaming, addressing diverse pedagogical dimensions.

Keywords: serious games; applied games; game technology; game development; software components; RAGE; performance; emotion; natural language processing; learning analytics

1. Introduction

The creation of games, be it leisure games or serious games, is a complex process of game design, programming, content production, and testing (Björk et al, 2006; De Gloria et al, 2014; Westera et al, 2008; Salen et al, 2004). To some extent the efforts, time and costs of game development can be reduced by reusing existing game objects that are available on various online portals such as the Unity Asset Store, Guru, Game Salads and the Unreal Marketplace. Unfortunately, most of the game objects offered come with three main drawbacks. First, most objects are media objects (e.g. terrains, audio, buildings, weapons, user-interface objects, and templates), which do reduce the efforts required for content creation, but still preserve the programming load. Second, the objects very rarely expose pedagogical affordances, while they do not discriminate between leisure games and serious games. Third, the scarce game software objects are bound to a particular game engine or platform and will not run on other platforms: a Unity3D software component will only run in the Unity3D engine.

The RAGE project, which is a Horizon 2020 research and innovation project of the European Commission, addresses these issues by making available a set of serious game software components that can be used across a wide variety of game engines, game platforms, and programming languages that game developers have in use (Van der Vegt et al, 2016a). Up to forty reusable software components are foreseen, which cover a wide range of functionalities particularly tuned to the pedagogy of serious gaming. The components address player data analytics, emotion recognition, stealth assessment, personalisation, game balancing, procedural animations, language analysis and synthesis, interactive storytelling, social gamification, and many other functions. The European Commission supports the RAGE project as it envisions a flourishing serious games industry that both stimulates the creation of jobs in the creative industry sectors and helps to address a variety of societal challenges in education, health, social cohesion, and citizenship. Also, the approach of open

software components complies with Europe´s Digital Single Market policy, which is among the top priorities of the European Commission: enhancing Europe's position as a world leader in the digital economy by supporting open standards, open data, and open access. Moreover, RAGE complies with the principles of fair competition as its portable open software components help to avoid game engine vendor lock-in, monopolies, and price-fixing.

In this paper we present a first selection of software components and explain their design and purpose. These include components for emotion detection, performance statistics, natural language processing and data storage. Before going into the details of these game software components, we will first outline the overall architecture.

2. The RAGE Component-Based System Architecture

The purpose of the RAGE component architecture is to allow for the easy integration of reusable software modules across a multitude of different game engines and platforms. The architecture combines a service-oriented architecture (SOA) using web-services for communication with remote agents via the http protocol (e.g. REST) and a client-side framework for components that are to be fully integrated into the game engine (Van der Vegt et al, 2016a; Van der Vegt et al, 2016b). Such a hybrid approach is needed to bypass the limitations of both SOA and client-side solutions. Generally, SOA offers several advantages such as decoupling from implementation details and a high degree of reusability of services. On the downside, SOA may reduce system performance due to frequent network calls and additional overheads, and it offers limited customisation and configuration of services by service consumers. Moreover, the player is supposed to be permanently online, which need not always be the case. The RAGE client architecture relies on a limited set of well-established software patterns and coding practices aimed at decoupling abstraction from its implementation. It avoids dependencies on external software frameworks and minimizes code that may hinder integration with game engine code. However, if client-side components require extensive processing, remote services may be a better option to avoid poor game performance on the local machine, e.g. reduced frame rate or reduced game responsiveness. Client components will primarily be developed in C# and JavaScript/TypeScript as to comply with most popular development platforms. Even so, proofs of concept specifically designed within RAGE are available in C++ and Java (Van der Vegt et al, 2016a). Figure 1 sketches the basic structure of the RAGE architecture (Van der Vegt et al, 2016a; Van der Vegt et al, 2016b).

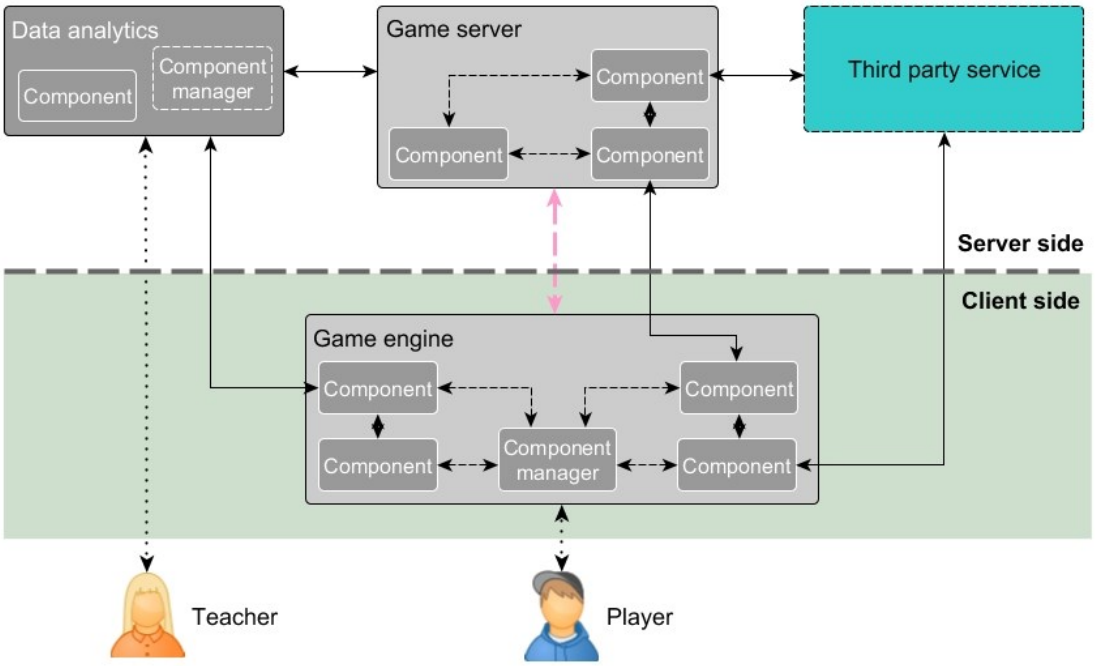

Figure 1. Outline of the RAGE component-based architecture and its diverse data communication routes.

In figure 1, the game engine and the eventual game server are in pale grey. Dark rectangles represent RAGE software components. The RAGE data analytics server is a central server-side system that aggregates interaction data from the population of players to be inspected by teachers and educators. This server handles authentication and authorisation, and acts as a gateway proxy server to underlying components and services.

It includes a component manager for the registration of web services. Client-side RAGE components are coordinated by a local component manager. RAGE components, either client-side or server-side, can directly interact with each other. Server-side components can be included in the game server or use a separate server. Third party web services can be called by both client-side and server-side components.

3. The Emotion Detection Component

Emotion is a highly relevant factor in both game play and learning (Bower, 1981; Butz et al, 2015). Emotions can strongly influence cognitive processes as they relate to memory and action (Pekrun, 1992). It is widely accepted that emotion and cognition represent highly interactive and integrated processes (Izard, 2009), a perspective consistent with the high degree of connectivity among the brain's neural structures and systems as established by neuroscience studies (Heraz et al, 2011). So far, however, the player´s emotions have been systematically neglected in the player models from serious games, due to difficulties and limitations of detection methods. Advances in machine learning and pattern recognition have now enabled a viable and affordable detection of emotions.

The RAGE Emotion Detection component enables game developers to easily include the player's emotions in the game play. This client-side component uses the player's webcam video stream for the real-time recognition of the player's emotions from facial expressions. The Emotion Detection component returns a value for one out of six basic emotions (happiness, sadness, surprise, fear, disgust, anger) suggested by Ekman et al (1978), as well as the neutral emotion. The returned emotional state values can be used in various ways to enhance the pedagogical quality of serious games. First, tracking the player's moods during gameplay can be used to improve game balancing and personalisation, for instance by informing pedagogical support interventions in the game, such as hints, guidance, and feedback. Second, the unobtrusive tracking of player emotions enables the evaluation and analysis of user experiences during gameplay; issues and bottlenecks in the game could be identified and removed. Third, when emotions are part of the learning content (e.g. in job interview training, negotiation training, or social skills training), the Emotion Detection component can be used for assessing the players' mastery of emotion expression and for providing feedback based on their performances. Figure 2 shows a screenshot of an applied game in communication training that provides direct feedback on the quality of expressed emotions (Bahreini et al, 2014).

Figure 2. Emotion-based feedback in a communication dialogue training game (Bahreini et al, 2014).

This RAGE component comes with several advantages. It provides a simple and low cost solution as it just uses a standard webcam. It offers a high accuracy and reliability that are comparable with those of expert human raters. In many respects the webcam-based emotion detection is superior to existing approaches such as physiological sensors and questionnaires, which are obtrusive, discontinuous, disturbing, or unreliable. A

developed and tested version of this component has been already reported in a previous study (Bahreini et al, 2014).
Technically, the component uses as an input the webcam's video stream with a frame rate of up to 30 frames per second. For each image received from the webcam, the component returns a probability table for the set of seven emotional states. No storage capacity for the video streams is required as these are processed on the fly and not stored for retrieval or post-processing. This component is being developed as a client-side component in both C# and JavaScript/Typescript. A first experiment with this component involves its use in a job interview game.

4. The Performance Statistics Component

Driven by the ever growing datasets in education and training, many educators and scientists consider learning analytics as a means of improving the quality of learning (Buckingham Shum et al, 2012). Serious games would be an excellent target of learning analytics as playing a serious game produces highly individualised data trails that reflect the player´s personal choices, behaviours, and performances. However, due to multiple constraints, it still is a complex case for teachers and game developers to use these trails in a productive way. First, the diversity and large amounts of logged data creates a challenge in terms of deciding which entries to select and which statistical methods to apply. Second, the correct application of statistical methods requires detailed knowledge about the procedures and their limitations. Third, the interpretation of statistical results may be deceptive due to hidden assumptions, constraints, and common pitfalls associated with statistics.
The RAGE Performance Statistics component provides teachers and game developers with a utility that overcomes these problems. Basically, the component requires relevant player performance data as inputs and then automatically generates a set of statistical indicators about the player's learning progress, supplemented with validity checks, key events, and guidelines for their interpretation. The performance statistics component processes two types of input metric that are most commonly used for indicating learning progress (Chance, 2009). The first metric reflects the time required to complete a task, while the second one is indicative of the performance on a task (in terms of errors made or points scored). Tasks are well-identified game activities with a clear start and ending, and possibly well-related to learning goals. Differences in progress may exist between learners as some learners may complete the task in one play-through, while others may need multiple trials. It is essential for tutors to have access to such data in order to be able to provide further assistance and feedback when needed.
The Performance Statistics component works on the server-side, enabling it to aggregate player data at population level. Valuable insights are gained from comparisons at population level, as it provides benchmarks for individual students. After game data is sent to the server, the Performance Statistics component performs the statistical analyses and presents the results through a web interface in either tabular or graphical form, including an interpretational message.
At individual level, each learner's score is compared to the group means and standard deviations, while notifications are triggered in case of substantial deviations. Also, a self-comparison is provided for each learner in case of multiple trials on the same task. Learning progress can be identified by fitting a regression curve to the learner scores or the time spent. A second level of analysis is the group level: the performances of different groups, e.g. schools, teams, or classes, are compared. The groups' progress over trials can be derived from individual player data. T-tests are used to identify differences between groups. In addition, the component supports a third level of analysis: by aggregating task performances across different trials learners can be mutually compared. Deviations from normality or from the average may signal problems. For instance, skewness of the distribution may indicate balancing issues with the task at hand. Similarly, large standard deviations across task performance may indicate unclear or inappropriate task definitions.
The Performance Statistics component offers a practicable suite of statistical options that may help teachers gain insight into the learning performances of their learners and that provide timely triggers to detect uneven progress. No substantial prior knowledge about statistics is required from the teacher or the developer to use this component. The performance statistics component is being developed in Java and JavaScript. A first implementation will be tested in a communications training game for Dutch vocational schools.

5. Language Technology Services

For a long time, computer-based learning tools were not capable of processing the learners' textual inputs; instead they had to rely on the unambiguous responses from multiple-choice items. Consequently, learners were required to recognise and select the correct answers from a limited set of options, rather than express

their thoughts using natural language. This makes a big difference as the active use of language is closely related to comprehension and in-depth cognitive processing. Obviously, language is a key component to education, because it represents the conduit for communicating and understanding information. In recent years, Natural Language Processing (NLP) techniques have become accurate, efficient, and reliable methods to analyse language and have gained a broader usage. So far, however, their application in education has been scarce and indirect (e.g. simple word count and checking for plagiarism). Given the advances in language technologies, the RAGE project will make available a set of NLP services to be directly integrated in serious games. These services facilitate a variety of improvements in serious game design, which all concern the direct analysis and interpretation of spoken or written traces of players while interacting with the game and/or with fellow players. In addition, our language services facilitate a wide range of educational scenarios covering, for example: the automated evaluation of summaries with regards to comprehension prediction, assessment of CVs and cover letters in terms of adequacy and optimism, providing personalised recommendations for improving one's writing style, and many more.

NLP techniques provide the ground for computational analyses focused on various aspects of language and are related to particular tasks or domains. The tools can measure a variety of linguistic features that are important for understanding texts and predicting learners' comprehension, including textual cohesion, textual complexity, and sentiment analysis, often referred to as opinion mining, which is identifying and extracting subjective information from texts. The services will be based on the *ReaderBench* framework (Dascalu, 2014; Dascalu et al, 2014), which introduces a generalised, multi-lingual, automated model applicable to both essay-like or story-like texts as well as conversations in multi-user games, Computer Supported Collaborative Learning (CSCL), or Communities of Practice (cf. figure 3).

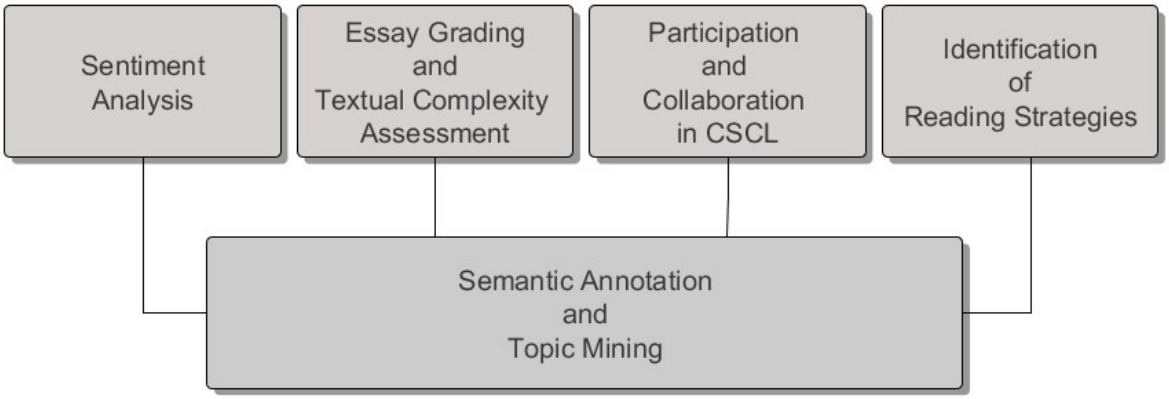

Figure 3. Global view of the ReaderBench NLP framework.

Basically, the framework allows for predicting and assessing comprehension, both for individual and collaborative learning in teams. While comprehension prediction evaluates the textual complexity or difficulty of learning materials, comprehension assessment refers to the a posteriori processing of learner productions, e.g. reports, answers, or contributions to (online) conversations. The core of our language analysis is cohesion, which can be viewed as the sum of lexical, grammatical, and semantic relationships that link together textual segments. Cohesion is automatically computed in terms of semantic distances in lexicalised ontologies (e.g. WordNet or other located instances), Latent Semantic Analysis (LSA) vector spaces and Latent Dirichlet Allocation (LDA) topic distributions (Dascalu, 2014). The two latter semantic models need to be trained on dedicated corpora based on language and domain specificities. Moreover, tailored natural language processing pipelines are enforced by a variety of methods for tokenising, splitting, selection of dictionary-only words, stop words elimination, stemming, part of speech tagging, dependency parsing, lemmatisation, and named-entity recognition (Dascalu, 2014).

Four services that have been integrated in the ReaderBench framework are explained below. First, sentiment analysis (opinion mining) allows for tracking the mood in terms of multiple valences that can be extracted, e.g. positive words versus negative words (cf. figure 3). In RAGE, sentiment analysis will be used for assessing the mood derived from application letters that learners have to write in a job application training game. Second, automated essay grading refers to the assessment of learners' texts in reports and responses to open questions. The service helps to identify specific flaws in writing, which can be improved (e.g., length of phrases, use of discourse connectors, lack of cohesion between adjacent phrases) and allows for providing personalised feedback to learners. Such feedback has been demonstrated to have a big positive impact on

writing style quality (Crossley et al, in press). The analysis of textual complexity includes a multitude of factors such as classic readability formulas, surface metrics commonly used in other automatic essay grading systems, syntax indices, as well as semantics and discourse. Importantly, our model is extensible, covers multiple languages (e.g., English, French, Romanian) and is centred on cohesion, introducing innovative indices, such as document cohesion flow (Crossley et al, in press) or Age of Exposure (Dascalu et al, 2016). Third, conversation analysis is applied to support tutors in the time-consuming process of analysing online conversations. Two complementary computational models for assessing participation and collaboration are applied. One model performs a longitudinal analysis of an ongoing conversation and accounts for collaboration from a social knowledge-building perspective. The second model takes a dialogical perspective, while it uses the alternate interactions of speakers for a transversal analysis of the conversation. Fourth, the use of reading strategies by learners during their self-explanations is widely recognised as a determinant of reading comprehension (McNamara, 2004). This RAGE component automatically identifies the employed strategies involving metacognition, causality, bridging, paraphrasing, and elaboration. The identified strategy profiles are reliable predictors of the individual comprehension level, while the accuracy of comprehension prediction can be significantly raised with the integration of textual complexity indices.
All components are exposed as REST web services that can be easily integrated in third-party custom applications that query the functionalities provided by the web server. The entire framework is implemented in Java and includes multiple libraries, out of which the most notable are Stanford Core NLP, Apache Mahout, Mallet, and Gephi.

6. The Shared Data Storage Component

Since a game might include multiple RAGE components that potentially use shared datasets, a coordinated data management service is needed. This service is provided by the Shared Data Storage component. It offers a client-side technical utility to component developers, as well as game developers, for defining and storing datasets without bothering about compliance with the overall RAGE architecture and the usage in different game engines (Van der Vegt et al, 2016). Simple examples of shared data include the player ID, the associated player scores, or emotion values resulting from the Emotion Detection component. The names and data types should be dictated by a centralised vocabulary. Various RAGE components define taxonomy-based data or data models that can be represented as tree structures (domain model, competency model, player model, etc.). The Shared Data Storage component offers a centralised function to manage and store such information without the need for data duplication and additional coding, thus reducing overheads of testing and maintenance, while preserving the portability to popular game development engines, such as Unity3D and Xamarin. The Shared Data Storage Component allows the model data to be stored in (and/or retrieved from) four different locations (cf. figure 4).

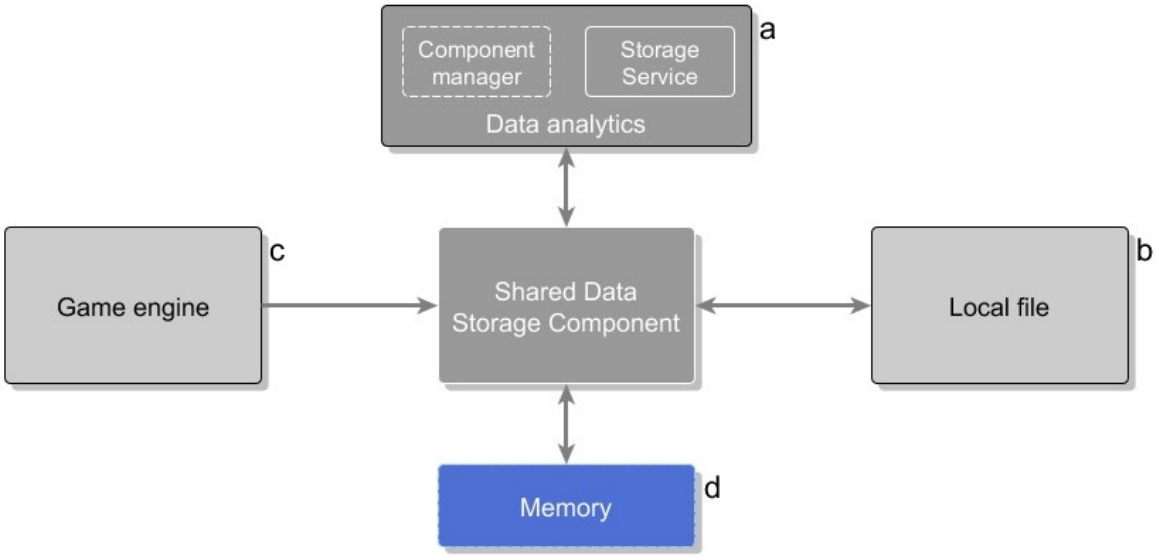

Figure 4. Persistence sources and destinations of the Shared Data Storage Component.

The supported locations are a) the Storage Service offered by the RAGE Proxy Server, b) the client machine's local filesystem, c) the game using the component, and d) transient data.

a) Online storage offered by the RAGE Proxy Server:
   This service is available after authentication against the Proxy Service. To use this service, the proxy should be up and running, and the player should be online as well.
b) Storage on the client's local filesystem:
   The Shared Data Storage Component's services include the ability to persist in the local storage when there is no online connection or when authentication against the proxy system fails.
c) Storage in the game:
   If the game is used as storage location, the game is queried for values whenever needed. This data is read-only for not interfering with the game. Example data would be the time spent within the game, current location, username, or user ID.
d) Transient data:
   The component also supports that parts of the data structure can be marked as transient and thereby excluded from persistent storage.

For accessing the various storage locations, the Shared Data Storage component uses bridge interfaces that are potentially shared with other RAGE components. Three data exchange formats are currently supported: XML, JSON (Crockford, 2006) and .Net Binary. Binary serialisation is used to store the structure in XML and JSON formats: binary data are converted into a textual format by base64-encoding. The Shared Data Storage component keeps developers away from a tedious serialisation process. Serialisation may look deceptively easy, but in fact can be complex for certain data types. Especially JSON is known for problems in inferring datatypes and might not be lossless when used out of the box. JSON serialisers, such as the popular Newtonsoft Json.Net library (Newtonsoft, 2016), often try to deduce the type from the data, whereby data types may change unexpectedly. RAGE's Shared Data Storage component, however, offers a lossless round-trip serialisation, meaning that structure, data and datatypes are identical after restoring. In order to achieve such lossless round-trip serialisation, type information is saved alongside the data. The component also allows for restoring the data from multiple sources as the data and the internal data structure are serialised and persisted separately. Altogether, the Shared Data Storage component offers developers the benefits of having to integrate only one component, while not having to duplicate code for lossless serialisation, persistence, and the authentication and communication with a storage server.

7. Discussion and Conclusion

This paper has presented four separate software components based on the RAGE applied gaming framework. The main purpose of the RAGE framework and its components is to support serious game developers in the process of game development. The components offer game developers easy access to advanced technologies that might otherwise be beyond reach. The components are available as reusable code that can be easily integrated in various existing game engines. Multiple RAGE components can be added to the same game, enhancing their overall support for the learning. More importantly, the usage of RAGE components allows for developing games easier, faster and more cost-effectively. Thereby, the RAGE project aims to accommodate the serious game industry and contribute to amplifying the serious game market. The four components presented in this paper are a first selection out of forty components that are anticipated. But, possibly even more important than the components themselves are the technical infrastructure and component methodology that RAGE will make available and that would enable third parties to create and publish their own RAGE-compliant software technologies. The infrastructure includes tools for adding RAGE metadata, quality assurance checkpoints and authoring widgets for component-related content, and it allows third parties to package, upload, and publish RAGE compliant components to the software repository.

Still, some limitations have to be taken into account. In particular, the unconstrained portability of client-side components is subjected to constraints (Van der Vegt et al, 2016a). First, for pragmatic rather than principle reasons, RAGE clients will only use code bases in C# (for desktop and mobile games) and JavaScript/TypeScript (for browser games), which are predominant products of compiled languages and interpreted languages, respectively. Implementing all components in three, six or even more programming languages would simply require too many resources. Second, testing the integration of components in game engines will necessarily be limited to a few major engines for the same reason. It is virtually impossible to cover the wide variety of game engines and platforms that are available (Wikipedia lists 169 different engines), not to speak about compatibility issues with different versions. Third, the inventory of RAGE components should not be conceived as a honey pot that would allow for unlimited snitching. The integration of a large number of components in a game may cause coordination or synchronisation problems, potentially leading to weak performance, malfunctioning, or ultimately system crashes. Although no fundamental problems have been traced so far, the

implementation of many components can only be tested on a case-by-case basis. Fourth, although the integration of components in game engines should not be that complex, some components may require extensive preparations before running properly. For instance, in AI components or some of the language technology components, models may need to be trained and tuned to be practicable in specific domains or educational contexts.

Notwithstanding these potential issues, RAGE's software components will work correctly in major game engines and on major platforms, and thereby the project contributes to seizing the potential of serious games for teaching, learning, and various other domains.

Acknowledgement

This research was partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 644187, the RAGE project (www.rageproject.eu).

References

Bahreini, K., Nadolski, R., and Westera, W. (2014) "Towards Multimodal Emotion Recognition in E-learning Environments", *Interactive Learning Environments*, 1-16, advance online publication. doi: 10.1080/10494820.2014.908927.

Björk, S. and Holopainen, J. (2005) *Patterns in Game Design*, Charles River Media, Hingham.

Bower, G. H. (1981) "Mood and Memory", *American Psychologist*, Vol. 36, pp 129-148.

Buckingham Shum, S. and Ferguson, R., (2012) "Social Learning Analytics", *Educational Technology & Society*, Vol. 15, No. 3, pp 3–26.

Butz, N., Stupnisky, R., and Pekrun, R. (2015) "Students' Emotions for Achievement and Technology Use in Synchronous Hybrid Graduate Programmes: A Control-Value Approach", *Journal of Research in Learning Technology*, Vol. 23, doi:10.3402/rlt.v23.26097. Retrieved from http://www.researchinlearningtechnology.net/index.php/rlt/article/view/26097.

Chance, P. (2009) *Learning and Behavior (6th edition)*, Belmont CA, Wadsworth, Cengage Learning.

Crockford, J. (2006) *The application/Json Media Type for JavaScript Object Notation (JSON)*, Fremont CA, Internet Engineering Task Force. Retrieved from http://www.ietf.org/rfc/rfc4627.txt.

Crossley, S.A., Allen, L.K., & McNamara, D.S. (2016, in press). The Writing Pal: A writing strategy tutor. In S. A. Crossley & D. S. McNamara (Eds.), *Handbook on Educational Technologies for Literacy*. New York, NY: Taylor & Francis, Routledge.

Crossley, S. A., Dascalu, M., Trausan-Matu, S., Allen, L. and McNamara, D. S. (2016, in press) "Document Cohesion Flow: Striving towards Coherence", in Proc. of the 38th Annual Meeting of the Cognitive Science Society. Philadelphia, PA: Cognitive Science Society.

Dascalu, M. (2014) "Analyzing discourse and text complexity for learning and collaborating", *Studies in Computational Intelligence* (Vol. 534). XIV, 279 pages. Cham, Switzerland: Springer.

Dascalu, M., Dessus, P., Bianco, M., Trausan-Matu, S. and Nardy, A. (2014) "Mining texts, learner productions and strategies with ReaderBench", in A. Peña-Ayala (Ed.), *Educational Data Mining: Applications and Trends* (pp. 335–377). Cham, Switzerland: Springer.

Dascalu, M., McNamara, D.S., Crossley, S.A. and Trausan-Matu, S. (2016) "Age of Exposure: A Model of Word Learning", in 30th AAAI Conference on Artificial Intelligence (pp. 2928–2934). Phoenix, AZ: AAAI Press.

De Gloria, A., Bellotti, F. and Berta, R. (2014) "Serious games for education and training", *International Journal of Serious Games*, Vol. 1, No. 1. Retrieved from http://journal.seriousgamessociety.org/.

Ekman, P., and Friesen, W. V. (1978) *Facial Action Coding System: Investigator's Guide,* Consulting Psychologists Press, Palo Alto.

Heraz, A., and Frasson, C. (2011) "Towards a Brain-Sensitive Intelligent Tutoring System: Detecting Emotions from Brainwaves," *Advances in Artificial Intelligence*, Vol. 2011, Article ID 384169, 13 pages, 2011. doi:10.1155/2011/384169. Retrieved from http://www.hindawi.com/journals/aai/2011/384169/.

Izard, C. E. (2009) "Emotion Theory and Research: Highlights, Unanswered Questions, and Emerging Issues", *Annual Review of Psychology*, Vol. 60, pp 1-25. doi:10.1146/annurev.psych.60.110707.163539. Retrieved from: www.ncbi.nlm.nih.gov/pmc/articles/PMC2723854/.

McNamara, D.S. (2004) "SERT: Self-Explanation Reading Training. *Discourse Processes", Vol. 38*, pp 1-30.

Pekrun, R. (1992) "The impact of emotions on learning and achievement: Towards a theory of cognitive/motivational mediators", *Journal of Applied Psychology*, Vol. 41, pp 359-376.

Newtonsoft Json.Net (2016) Json software framework available at http://www.newtonsoft.com/json.

Salen, K. and Zimmerman, E. (2004) *Rules of Play, Game Design Fundamentals*, MIT Press, Cambridge, MA.

Van der Vegt, W., Nyamsuren, E. and Westera, W. (2016b) "RAGE reusable game software components and their integration into serious game engines", in: Georgia M. Kapitsaki and Eduardo Santana de Almeida (Eds*.), Bridging with Social-Awareness*, 15th International Conference, ICSR 2016, Limassol, Cyprus, June 5-7, 2016, Proceedings, Lecture Notes in Computer Science, Volume 9679 2016, pp. 165-180.

Van der Vegt, W., Westera, W., Nyamsuren, E., Georgiev, A. and Martínez Ortiz, I. (2016a) "RAGE Architecture for Reusable Serious Gaming Technology Components", *International Journal of Computer Games Technology*, advance online publication. doi:10.1155/2016/5680526. Retrieved from http://www.hindawi.com/journals/ijcgt/2016/5680526/.

Westera, W., Nadolski, N., Hummel, H. and Wopereis, I. (2008) "Serious Games for Higher Education: a Framework for Reducing Design Complexity", *Journal of Computer-Assisted Learning*, Vol. 24, No. 5, pp 420-432.