The RAGE Software Asset Model and Metadata Model

A. Georgiev[1], A. Grigorov[1,6], B. Bontchev[1], P. Boytchev[1], K. Stefanov[1*], K. Bahreini[2], E. Nyamsuren[2], W. van der Vegt[2], W. Westera[2], R. Prada[3], Paul Hollins[4], Pablo Moreno[5]

[1] Sofia University "St. Kliment Ohridski", Faculty of Mathematics and Informatics, Bulgaria
{atanas, alexander.grigorov, bontchev, boytchev, stefanov}@fmi.uni-sofia.bg
[2] Open University of the Netherlands, The Netherlands
{kiavash.bahreini, enkhbold.nyamsuren, Wim.vanderVegt, Wim.Westera }@ou.nl
[3] University of Lisbon, Portugal
rui.prada@tecnico.ulisboa.pt
[4] The University of Bolton, UK
pah1@bolton.ac.uk
[5] Universidad Complutense de Madrid, Spain
pablom@fdi.ucm.es
[6] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Bulgaria
grigorov@math.bas.bg

Abstract. Software assets are key output of the RAGE project and they can be used by applied game developers to enhance the pedagogical and educational value of their games. These software assets cover a broad spectrum of functionalities – from player analytics including emotion detection to intelligent adaptation and social gamification. In order to facilitate integration and interoperability, all of these assets adhere to a common model, which describes their properties through a set of metadata. In this paper the RAGE asset model and asset metadata model is presented, capturing the detail of assets and their potential usage within three distinct dimensions – technological, gaming and pedagogical. The paper highlights key issues and challenges in constructing the RAGE asset and asset metadata model and details the process and design of a flexible metadata editor that facilitates both adaptation and improvement of the asset metadata model.

Keywords: serious games, software assets, game assets, asset model, asset metadata model, metadata editor, gamification.

1 Introduction

In accordance with the requirements of the European Horizon 2020 Research and Innovation Program, the RAGE project[1] aims to develop and provide open advanced technology modules (assets) for applied gaming and to make these assets available

[1] http://rageproject.eu/

through a repository that encourages further development, sharing reuse and repurposing of the assets. The assets address pedagogically oriented functions that support game-based learning, particularly in the capture and assessment of user data and the support of strategic interventions and social representations in the game. The purpose of these assets is to support game studios in developing high-quality, pedagogically authentic applied games.

RAGE game assets will be stored in the game asset repository, a central component of a RAGE Applied Gaming Ecosystem. In this context many of the existing approaches and methodologies used by game development companies cannot be directly applied as they typically focus on design and development of bespoke domain-specific games.

In this paper we try to provide answer to the following main research problem, identified in the RAGE project: How do we enrich and transform advanced gaming technologies into self-contained assets for applied gaming that facilitate essential pedagogical functions, that can be linked together into higher level aggregates and that can be easily integrated in existing game platforms?

2 The RAGE Asset definition and model

A RAGE Asset is defined as a self-contained solution that demonstrates economic value potential, based on advanced technologies related to computer games, and intended to be reused or repurposed across a variety of game platforms. The RAGE assets comply with the asset definition of the W3C ADMS Working Group [19], which refers to abstract entities that reflect some "intellectual content independent of their physical embodiments". In principle, not all assets include software, e.g. media assets, common in game development, may refer to graphical objects, audio files, videos and other such objects. This paper focuses specifically on the software assets. The RAGE assets contain advanced game technology components (software), enriched and transformed to support applied games development. They are supported by value adding services and attributes (artefacts), such as instructions, tutorials, examples and best practices, instructional design guidelines and methodologies, connectors to major game development platforms and content authoring tools/widgets for game content creation. These additional artefacts not only support but enhance assets usage. Fig. 1 provides an exemplar of a RAGE asset and related artefacts.

In order to preserve the software asset's portability across different game engines and platforms, a component-based asset architecture has been described and validated elsewhere [1]. This architecture addresses both the internal workings of an asset and the level of interaction of assets with the outside world, including the mutual communications between assets. The RAGE architecture avoids any dependencies on external software frameworks and minimize any code that may hinder integration with game engines. Furthermore, it relies on a limited set of standard software patterns and well-established coding practices.

A centralized Asset Manager component is included as a coordinating agent, which is used for the registration of the assets and for the use of shared code that is commonly used by multiple assets, such as the name and the type of the game engine, or user login/logout info for assets that would need a user model.

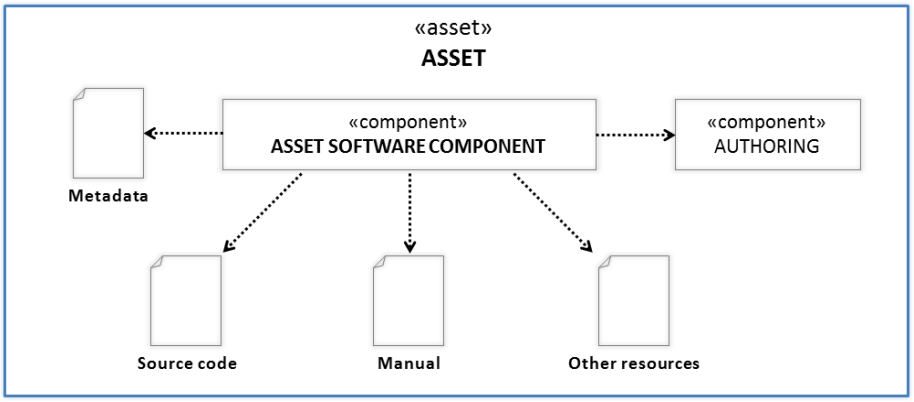

Fig. 1. Example of a RAGE Asset

Proofs of concept in four principal code bases (C#, Java, C++ and TypeScript/JavaScript) have validated the RAGE architecture [1]. In addition, the easy integration of the HAT asset (difficulty adaptation routine) in three different game engines (Unity, MonoGame and Xamarin) has been successfully implemented [2], demonstrating the portability and flexibility of the RAGE asset architecture.

Using interoperability components or named assets in game development involves three different broad disciplines from computer science: software engineering (because the assets are implemented in games as software components), game development (as the assets are integral elements of different games) and Technology-Enhanced Learning (TEL) (as the assets should provide additional pedagogic "values" to the games).

In the following paragraphs we present the principal outcomes across these three related disciplines related to our research (section Related work), further we define the RAGE Asset Metadata model, and finally discuss our initial implementation findings in relation to the research question, highlighting key issues to be addressed.

3 Related work

In the gaming domain the term *asset* is often applied to media files to be incorporated in a game. The Intel® XDK HTML5 Cross-platform Development Tool[2] offers an asset manager for game development in conjunction with several game platforms. Here assets are considered as audio-visual game objects, to be included in a project. Similarly, the Unity Asset Store[3], which is a prosumer-based community and market for game assets, is dominated by such media assets, these are imported and used in the Unity game engine. Historically game developers would apply the term *asset* for such media files rather as opposed to software artefacts, the Unity Asset Store is increasingly including software assets, e.g. code for physics, special effects, controller software, Graphical User Interface software, Artificial Intelligence, and maze generation.

[2] https://software.intel.com/en-us/html5/tools

[3] https://www.assetstore.unity3d.com

However, this particular and quite specific definition of assets in game world is not useful for the RAGE project purpose. For a more detailed understanding of how to describe and classify assets, we start with the analysis of best practices from the Technology-Enhanced Learning domain (TEL) and the games for learning (applied gaming and serious gaming) domain.

By exploring a variety of game ontologies/taxonomies, we have an established position on classifying assets as game components. The best suited to the RAGE project are from two related propositions: SharpLudus Game Ontology [5] as well as the Game-Based Learning Systems (GBLS) Ontology initiative [6]. They both define the main structure and functions of applied games and their relationships with the TEL domain. One of the directions in applied games research is linked with the mapping of game mechanics and learning activities [7]. By exploring the LM-GM model (Learning Mechanics – Game Mechanics), the study addresses the similarities between game mechanics and educational components at the implementation level. A simplified game model based on Bloom's theory is in a similar vein. Peeters [8] present an ontology related to applied games, consisting of 6 main areas: Task domain, Trainee, Didactics, Instructor, World, and System. A mapping between game activities, learning activities and learning resources was presented by Prensky [9].

Very close to these ideas is an approach based on the concept of educational game design patterns. In general, applied games design patterns should be based on the well-established knowledge of game design patterns [10], with the addition of a second foundation besides entertainment: pedagogy. A major conceptual tool for better applied game design is the definition of pedagogically informed game design patterns. Kiili [11] identified a number of patterns, proposing six categories addressing key educational aspects. The mapping of learning functions onto game design patterns was presented by Kiili [12].

Whilst these efforts have attempted to link game design with learning design, another approach is based on extending the IEEE Learning Object Metadata (LOM) standard with additional features reflecting the game functionalities and how they affect learning. The SG-LOM profile adds new fields to the LOM categories “Educational”, “Annotation” and “Classification” [13]. Hendrix [14] proposed a metadata schema for describing applied games also as an extension of the IEEE LOM standard by supplementing the number of fields to IEEE LOM and having two different levels. In [15] applied games are regarded as active learning objects (LO) that exchange information with the host Learning Management System (LMS) for tracking and assessment purposes.

An asset in the Information Technology (IT) domain of is generally defined as a collection of related artefacts that provides a solution to a problem [16]. Some asset definitions are restricted to content and/or media rather than software. For instance, Niekerk [17] distinguishes three major groups of “digital assets”: textual assets (digital assets), images (media assets), and multimedia assets (a combination of different content forms). The IMS content packaging information model [18] likewise uses the word asset to describe the term resources: “the resources described in the manifest are assets such as Web pages, media files, text files, assessment objects or other pieces of data in file form”.

IBM´s Reusable Asset Specification (RAS) [16] uses a high level definition of an asset as “a collection of related artefacts that provides a solution to a problem”, allowing

it to package together as an asset almost anything: Models, Design documents, Patterns, Web services, Frameworks, Components, Requirements documents, Test plans, Test scripts, Deployment descriptors, Model templates, UML profiles, Domain specific languages, etc. The W3C specification of the Asset Description Metadata Schema [19] defines an asset as an abstract entity that reflects some intellectual content. An asset differs from an asset distribution, which is typically a downloadable computer file (but in principle it could also be a paper based document or API response) that implements the intellectual content of an Asset.

A model-driven serious games development framework is defined by [20]. They present a platform-independent model incorporating nine core units, namely user interfaces, models of game content, game technology and game software, Model Driven Engineering (MDE) tools, components library (involves art assets, artificial intelligence, physics and e-learning sites), code templates, artefacts, technology platform, operating platform and software. The implementation of this model was presented in [21].

4 RAGE Asset Metadata Model

Metadata is an essential part of the information infrastructure and is critical for creating information services including description, classification, organization, store, search, creation, modification and aggregation of information [3]. Metadata models define the essential characteristics of information assets [4] describing their key components and functions. These models influence and support key services required for asset management including cataloguing, workflow to create and store the assets, how to use, reuse and repurpose assets, etc. Additionally when users or processes interact with the assets, this interaction takes place within the metadata model framework.

The RAGE asset metadata refer to machine-readable information such as keywords and semantic information that can be used by the repository's search engine. Metadata also include information that is essential for running the asset software in an operational environment, e.g. on a game platform. The metadata includes version information and data about dependencies from other software assets. Consequently, the definition of the RAGE Asset metadata model is a key element in the RAGE project, which enables authentic implementation of the RAGE Ecosystem for the development and exchange of RAGE assets.

Before defining the metadata model, we performed an extensive needs assessment study [24], including asset developers, educators and game producer. In order to support identified set of services through the software repository and other related tools and, in parallel, to be close to the specified domain of reusable gaming components (RAGE software assets), the RAGE metadata model is focused on the following main aspects:

- **Technical** – how the RAGE asset might be used by game developers. We follow the RAGE asset model which describes assets as software components.
- **Contextual classification** – here we focus primarily on the pedagogical, educational and game characteristics, whilst leaving space for further characteristics.
- **Usage** – not restricted to how to install and configure the software, but also providing additional artefacts such as training materials, tutorials, educational goals, etc.

- **Intellectual Property Rights** – to enable various business models proposed by RAGE project to be implemented.

Whilst designing the model, we broadly adhered to general metadata design principles as specified in [3]:

- **Reusability** – reusing where possible existing metadata models, standards and available taxonomies and ontologies. We reuse parts of the RAS and ADMS metadata fields as well as parts of the LOM taxonomy.
- **Flexibility** – to easily facilitate extension of the metadata description of an asset with additional features and characteristics.
- **Simplicity** – we define most of the fields as not mandatory in order to easy the efforts of asset developers. We plan to develop tools for automatic extraction of the most important metadata field values.

The RAGE asset metadata model reuses and extends the specifications of RAS [16] and ADMS [19]. We have chosen the approach to use a core subset of RAS (see elements Asset, Solution, Usage, Artefacts, Requirements, Design, Implementation, Tests from Table 1. Description of RAGE metadata schema elements) and extend it with elements from ADMS (like Classification, Context, Concepts), IEEE LOM (used inside Classification and Context) and metadata related to the applied games domain. We could not use LOM (or even SG-LOM) directly, as it does not provide features for describing compound software objects. The most comprehensive and close to our needs RAS model is too general and complex, has slow adoption and is difficult for users [13, 15, 16]. It is also not consistent with current Software Engineering (SE) practices, as it supports the "waterflow" model for software development. For this reason we simplify significantly elements reused from RAS, and changed some details. The ADMS does not provide support for external artefacts and, similar to LOM, is lacking support for important SE features.

Several taxonomies are used as different Conceptual schemes inside the Context for describing educational elements such as Learning Goals, Knowledge transfer, Skills, Educational Disciplines, Teaching Phase, Learning Purpose, Educational Context, etc.

Although the internal representation of the RAGE metadata in the asset repository is in RDF, we have chosen to use XML as a manifest file format (a special file that contains information about the files packaged in a RAGE asset package) and an XML schema for the model for the following reasons:

- If the manifest file in the asset package is in RDF, it is difficult to validate it – as demonstrated in [22].
- The URIs of the asset and artefacts are automatically generated after the asset is ingested in the repository so we cannot use them in the manifest beforehand.

Thus, we have chosen to use an XML schema for validation of the manifest files and to follow the approach used by the Europeana[4] project for representing RDF in

[4] http://www.europeana.eu/portal/

XML. We have used some of the XSD schema files used by Europeana[5] and modified them according to the specifics of the RAGE project. This approach provides for automatic validation of manifest files and easy transformation from XML to RDF and vice versa.

The RAGE Metadata Model defines the format of asset metadata as an XML schema, which is implemented in all tools that require the processing of these metadata, e.g. a package metadata editor, the asset repository, asset installation widgets, etc.

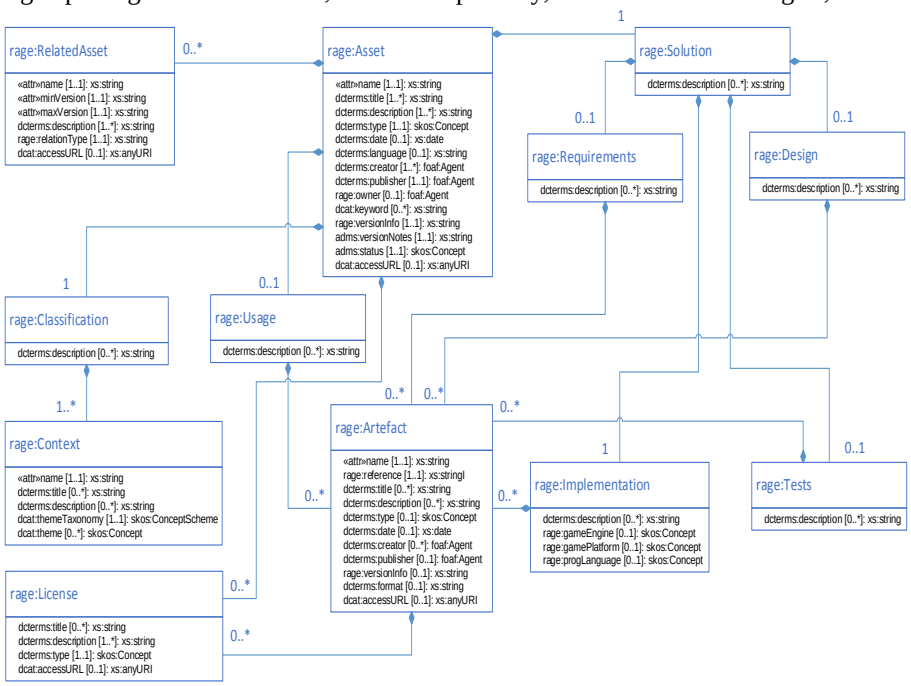

Fig. 2. The RAGE metadata schema

The XML schema represented by a UML class diagram in Fig. 2 can be interpreted while taking into account the following issues [23]:

- A UML class represents a complex XML element. For example, Asset is a complex XML element that has XML attributes and/or child elements.
- Within a UML class definition, the prefix «attr» denotes that the corresponding field is an XML attribute (and not a child element).
- Within a UML class definition, a field without «attr» prefix denotes a child element nested inside the element represented by the UML class definition. The field represents either a new definition of a simple element or a reference to an element (either simple or complex) defined in a referenced schema (e.g., dcterms).

[5] https://github.com/europeana/corelib/tree/master/corelib-edm-definitions/src/main/resources/eu

- Composition connection denotes a parent-child relationship between two complex elements defined in the RAGE Metadata Schema (RMS).

Table 1. Description of RAGE metadata schema elements provides a textual description of the XML elements of the RAGE metadata schema.

Table 1. **Description of RAGE metadata schema elements**

Asset	A self-contained solution that demonstrates economic value potential, based on advanced technologies related to computer games, and intended to be reused or repurposed in a variety of game platforms and scenarios.
Classification	Includes a set of descriptors for pedagogical classification of the asset (learning goal, learning functions, educational patterns used, etc.) as a learning object, as well as a description of the educational context(s) for which the asset is relevant
Context	Defines a conceptual frame, which helps explain the meaning of elements in the asset (game context where asset can be used, educational context, and links between game and learning mechanics in the game).
Concept Scheme	A vocabulary, thesaurus or taxonomy used for organizing concepts.
Concept	Represents a particular concept within a vocabulary, thesaurus or taxonomy.
Solution	Describes the artefacts of the asset.
Usage	Contains information for installing, customizing, and using the asset.
Related Assets	Describes the asset's relationship to other assets.
Artefact	Any physical element of an asset corresponding to a file on a file system. Artefacts can include also version and license information.
Requirements	Contains artefacts that specify the asset requirements such as models, use-cases, or diagrams.
Design	Contains artefacts that specify the asset design such as diagrams, models, interface specifications, etc.
Implementation	Has a collection of artefacts that identify the binary and other files that provide the implementation.
Tests	Contains artefacts (models, diagrams, artefacts, and so on) that are intended to describe the testing of the asset such as testing procedures, concerns and test units.
License	Contains conditions or restrictions that apply to the use of an asset or artefact, like is it in the public domain, can it be used for non-commercial purposes, etc.
Agent	Describes a person or organization that is a contributor (creator, publisher, and owner) of an asset or artefact.

5 The RAGE Metadata Editor

An important aspect of the metadata usability is the development of a metadata editor. This is a tool that facilitates the modification of asset metadata. The design of the RAGE metadata editor follows principles similar to these of the RAGE asset metadata model design in that:

- **Simplicity** – the editor hides the internal complexity of the metadata representation;
- **Flexibility** – the interface is generated on-the-fly, based on the metadata schema;
- **Usability** – various features for increasing the user comfort while editing metadata.

The visual simplicity is a principle that focuses on the user experience with the metadata editor. The metadata schema uses 7 different metadata structures that capture the spectrum of various data types and relations of asset descriptions. For example, there are data which are stored as attributes, as simple data entities or as sequences of

data. There are data which are conditional, or which have values from some predefined vocabulary. The editor hides this complexity and the user is not exposed to the internal structure of the metadata.

Fig. 3 represents a small fragment from a sample asset's metadata. It demonstrates how four of the different internal types are visualized consistently. For example, the *Name* of an asset, marked by (1), is internally stored as an XML tag attribute; the *Title* (2) is a simple string datum, the *Creator* (3) is an optional field, which is currently set to *Organization* and the elements *Name*, *Mbox* and *Homepage* (4) of the creator form a sequence of data, describing the creator's organization.

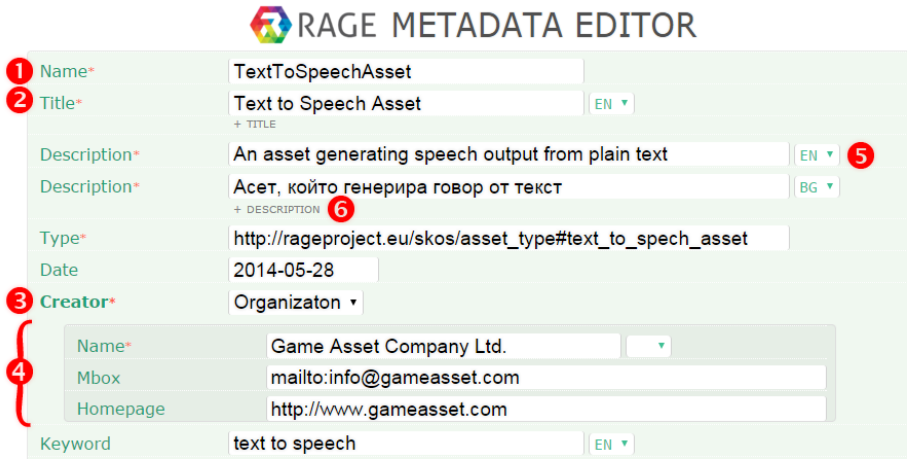

Fig. 3. Sample asset metadata visualized by the editor

Each metadata entry could be defined with a set of additional properties, like cardinality, current language, etc. They are also presented in the user interface. The small list boxes to the right of some metadata fields, like *Description* (5), allow the user to set the language of the content. The *Description* has cardinality 1+, which means that it is compulsory (marked by red asterisks) and it may have several instances (marked by the small +DESCRIPTION (6) buttons below the metadata box).

The principle of flexibility means that the editor is not bound to a fixed metadata structure. Instead, it reads the metadata description, extracts the schema file and then recursively processes schemas until it reconstructs the full structure of the metadata. Currently, except for the main asset metadata schema (stored in DefaultProfile.xsd), the editor also processes ADMS (Europeana's Asset Description Metadata Schema), DCAT (Data Catalog Vocabulary), DCTerms (Dublin Core Terms), FOAF (Friend-of-a-friend Schema) and RDF (Resource Description Framework).

Once the editor is familiar with the metadata structure, it builds the corresponding interface. Then it populates it with the actual metadata of the asset. This may recursively request the editor to generate new sections of the interface, if there are many instances of metadata with 0+ or 1+ cardinality. Because the actual structure could easily become too complex, the editor arranges the user interface elements in collapsible nested blocks.

The main advantages of a dynamic metadata editor are: (1) changes in the structure of RAGE asset metadata do not require corresponding changes in the RAGE metadata

editor; this allows the RAGE project the flexibility to adapt and improve the metadata model with minimal impact on other project software; and (2) the editor builds the interface by examining the schemas referenced by the metadata. As a result, it is possible to feed the editor with another metadata and other schemas. In this way the asset metadata editor can be used as a metadata editor of other RAGE entities, such as asset packages and artefacts.

When the interface is completely collapsed, it fits into a single screen and displays the most basic metadata of an asset like its name, description, keywords, versions, etc.

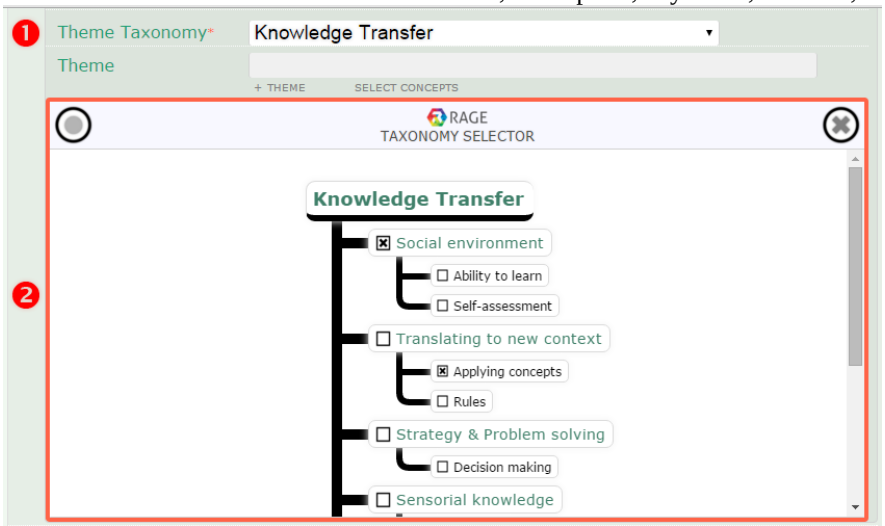

Fig. 4. The RAGE Taxonomy Selector used within the RAGE Metadata Editor

The classification of assets is implemented as references to taxonomies' concepts. To provide a convenient interface to concept selection the Metadata Editor embeds the RAGE Taxonomy Selector. This selector provides an interactive approach to select items from RAGE taxonomies that can be used to classify assets. Fig. 4 shows the Taxonomy Selector embedded in the Metadata Editor. The currently selected taxonomy is *Knowledge Transfer* (1) and the selector is (2).

After a user completes the editing of asset metadata, the editor generates an XML file. Prefixes are used in order to make XML tags shorter and easier to comprehend. Tags are aligned horizontally and vertically to provide clues for the hierarchy of metadata sections. Such styling easies the manual metadata manipulation during the initial phases of the software development.

6 Conclusion

In this paper we discussed a new approach for providing reusability, repurposing and interoperability of game software components called assets. The RAGE asset model is introduced, and the problem of how to describe applied game software assets in a machine-readable way to be used by game developers to add pedagogical and educational value to their games is discussed.

The paper presents in detail two key outputs for the RAGE project – the RAGE metadata model and the RAGE metadata editor. Both are created using simple design principles providing flexibility and reusability. The RAGE metadata model extends a core subset of RAS with elements from ADMS, IEEE LOM and applied games metadata. Its metadata schema provides support for the metadata of game artefacts, software engineering features and learning objects, this makes it very promising and useful for any applications dealing with applied gaming assets. The proposed interface of the metadata editor is driven by the RAGE metadata schema, but any structural changes in the RAGE asset metadata will not incur further changes in the metadata editor. Moreover, its interface changes automatically when feeding the editor with other metadata schemas. This makes it useful for editing other metadata such as these of asset packages or artefacts.

Both the RAGE metadata model and RAGE metadata editor provide a basis for classification, search and retrieval of gaming assets, by using references to taxonomies' concepts through a Taxonomy Selector. The generated XML file represents structurally the asset metadata and facilitates both software interoperability in game development and usage of assets' educational values for technology-enhanced learning applications.

Acknowledgements. This work has been partially funded by the EC H2020 project RAGE (Realising an Applied Gaming Eco-System); http://www.rageproject.eu/; Grant agreement No 644187.

References

1.van der Vegt, G.W., Westera, W., Nyamsuren, N., Georgiev, A., Martinez Ortiz, I.: RAGE architecture for reusable serious gaming technology components, International Journal of Computer Games Technology, Vol 2016 (2016), http://dx.doi.org/10.1155/2016/5680526

2.van der Vegt, G.W., Nyamsuren, N., Westera, W.: RAGE reusable game software components and their integration into serious game engines, accepted in the proc. Of the 15th International Conference on Software Reuse (ICSR-16) (2016).

3.Duval, E., Hodgins, W., Sutton, S., Weibel, S. L.: Metadata principles and practicalities. D-lib Magazine, Vol 8(4), DOI: 10.1045/april2002-weibel (2002).

4.Windsor, R.: Metadata Models: What They Are And Why You Need One For Successful Digital Asset Management, WebDAM white paper (2014) https://webdam.com/blog/what-are-metadata-models-pt-1/

5.Furtado, Andre: SHARPLUDUS: Improving game development experience through software factories and domain-specific languages, MSc Thesis, University of Pernambuco, Brasil (2006).

6.Raies, K., Khemaja, M.: Towards gameplay ontology for game based learning system design process monitoring, in TEEM '14 Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality, ACM New York, USA, pp. 255-260 (2014)

7.Arnab, S., Lim, T., Carvalho, M. B., Bellotti, F., de Freitas, S., Louchart, S., Suttie, N., Berta, R., De Gloria, A.: Mapping learning and game mechanics for serious games analysis. British Journal of Educational Technology. Special Issue: Teacher-led Inquiry and Learning Design. Volume 46, Issue 2, pp 391–411 (2015).

8. Peeters, M., Van den Bosch, K., Meijer, J.-J.Ch., Neerincx, M.A.: An Ontology for Integrating Didactics into a Serious Training Game. Proc. of the 1st Int. Workshop on Pedagogically-driven Serious Games (PDSG 2012), S. Bocconi, R. Klamma, and Y.S. Bachvarova (eds), CEUR Workshop Proceedings, volume 898, CEUR, Aachen, ISSN 1613-0073, pp. 1-10 (2012)
9. Prensky, M.: Digital Game-Based Learning, McGraw-Hill (2001).
10. Björk, S., Holopainen, J.: Patterns in Game Design, Charles River Media, Boston, MA (2004).
11. Kiili, K.: Call for learning-game design patterns. Chapter in the book: Edvardsen, F. & Kulle, H. (eds.). Educational games: design, learning and applications, Nova Publishers (2010).
12. Kiili K., T. Lainema, S. de Freitas, S. Arnab: Flow framework for analyzing the quality of educational games, Entertainment Computing 5 (4), pp. 367-377 (2014).
13. El Borji, Y., Khaldi, M.: An IEEE LOM Application Profile to Describe Serious Games «SG-LOM», International Journal of Computer Applications, Volume 86 – No 13, DOI 10.5120/15042-3404 (2014).
14. Hendrix, M., Protopsaltis, A., Rolland, C., Dunwell, I., de Freitas, S., Arnab, S., Petridis, P., et al.: Defining a Metadata Schema for Serious Games as Learning Objects. In the Proc. of the Fourth International Conference on Mobile, Hybrid, and On-line Learning eLmL 2012, pp. 14–19 (2012)
15. Torrente, J., Moreno-Ger, P., Martínez-Ortiz, I., Fernandez-Manjon, B.: Integration and Deployment of Educational Games, in e-Learning Environments: The Learning Object Model Meets Educational Gaming. Educational Technology & Society, 12 (4), pp. 359–371 (2009).
16. Ackerman, L., Elder, P., Busch, CV., Lopez-Mancisidor, A., Kimura, J., Balaji, N.A.: Strategic reuse with asset-based development, IBM RedBooks (2008).
17. van Niekerk, A. J.: The Strategic Management of Media Assets; a Methodological Approach. Allied Academies, New Orleans Congress (2006).
18. IMS-CP: IMS Content Packaging Specification (2004) https://www.imsglobal.org/ content/packaging/index.html.
19. ADMS: Asset Description Metadata Schema (ADMS), W3C proposal standard (2013) http://www.w3.org/TR/vocab-adms/
20. Tang, S., Hanneghan, M.: A Model-Driven Framework to Support Development of Serious Games for Game-based Learning. Paper presented at the 3rd Int. Conf. on Developments in e-Systems Engineering (DESE2010), London, UK (2010).
21. Tang, S., Hanneghan, M., Carter, C.: A Platform Independent Game Technology Model for Model Driven Serious Games Development, The Electronic Journal of e-Learning, Volume 11, Issue 1, pp.61-79 (2013)
22. Isaac, A.: Europeana and RDF data validation, Position paper from W3C RDF Validation Workshop, 10-12 Sept. 2013, https://www.w3.org/2001/sw/wiki/images/9/9b/ RDFVal_Isaac.pdf
23. Fowler, M., Scott, K.: UML distilled, Second edition. A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Longman Publishing Co., Inc., Boston (1999).
24. Saveski, G. L., Westera, W., Yuan, L., Hollins, P., Fernández Manjón, B., Moreno Ger, P., & Stefanov, K.: What serious game studios want from ICT research: identifying developers' needs. In: A. De Gloria and R. Veltkamp (Eds.), Proceedings of the GALA 2015 Conference, December 7-8, Rome, Italy, LNCS 9599, pp. 1–10. DOI: 10.1007/978-3-319-40216-1_4