

DESIGN AND DEVELOPMENT OF METADATA EDITORS: DATA-CENTRIC AND USER-CENTRIC APPROACHES

Pavel Boytchev

ABSTRACT. This paper presents the design and the development of two metadata editors for the RAGE project. The work is based on two conceptually different approaches. One of the editors follows a data-centric approach, while the other follows a user-centric one. Discussed are the design challenges, the internal structure and a comparison of both metadata editors.

1. Introduction. The EU Horizon 2020 Framework Programme for Research and Innovation spans over a period of seven years and is focused on “taking great ideas from the lab to the market” [1]. One of the projects funded by Horizon 2020 is RAGE — Realising an Applied Gaming Ecosystem [2]. The goal of this project is to support applied gaming industry by developing gaming modules, called *software assets* or just *assets*. These modules implement pedagogical functions, like capturing and assessing users and their learning via applied games [3]. The RAGE software assets contain a software

ACM Computing Classification System (1998): H.5.2, H.3.5, J.1, D.2.8.

Key words: metadata, metadata editor, metadata meta-editor.

core, implementing the functionality of the asset, as well as a rich set of complementary content (tutorials, examples, methodologies, etc.).

All assets are described by additional data, called metadata. These metadata follow a metadata model developed specifically for RAGE [4]. Currently the model envisions a significant amount of metadata to be collected for each asset. These metadata are used not only to describe the asset, but also to classify it and provide a convenient way for searching and filtering.

In order to programmatically analyse and process these metadata, they must be presented in a fixed format. However, due to the amount of metadata for an asset, it is not feasible for a human to work directly with this format — users would need a metadata editor, which is a software tool for presenting and manipulating metadata for people.

There are several approaches for making a metadata editor, depending on the goals, the target users and the actual metadata. Two of these approaches are used in the RAGE project. One of them is *data-centric* — i. e., the most important element is the metadata and all other elements, including the user, are adjusted to accomplish more effective metadata processing. This is the first approach used to make a metadata editor. The resulting software is called RAGE Metadata Editor and details about it are included in Section 2.

The other approach is *user-centric*. This approach considers the user as the most important element and all other elements are processed in a way to make the user's experience more comfortable and confusion-free. This is the other approach used to make a metadata editor for RAGE. To distinguish it from the previous editor, the second one is named *RAGE Metadata Wizard*. Details about it are presented in Section 3.

2. The data-centric approach. The RAGE Metadata Editor is software for editing metadata. Formally, it is a metadata meta-editor for RAGE assets, packages and other entities. It is designed to simplify the user interface visually and to adhere to some flexibility in metadata structure. The RAGE asset metadata entry form is split into several tabs, which use standard web controls for user input — text boxes, list boxes, buttons and links. The visual appearance of the form is customizable through widgets and CSS styling.

Internally, the meta-editor collects descriptions of the metadata from several schema files by recognizing several XML Schema structures. Then it dynamically generates a specific metadata editor for a given metadata set. The generated user interface depends on the structure of the metadata and the collected descriptions. The interface uses customizable widgets to present appropriate visual layouts for different metadata. When the editing is completed, the editor performs basic verification of the metadata and converts it back into the same format as the input metadata.

The concept of generating a metadata editor from a metadata schema is not new. MDEdit is an example of such editor [5]. Although it uses the same general concept as the RAGE meta-editor, two implementation aspects make it different: (1) MDEdit does not handle nested structures, which are required for RAGE metadata; and (2) the schema files are extended with new formatting tags, while RAGE relies on unmodified XSD schemas. Another editor is EUOSME (European Open Source Metadata Editor), which is not a production tool, but a research outcome. Its interface is close to the RAGE meta-editor, but it is tuned to geospatial and other metadata, which are not part of the RAGE metadata model [6]. The OLR3-Editor is a promising editor, which is described as able to accommodate any metadata description. The downside is that it “assumes only local data, which are stored in the OLR3 database, and is not yet adapted for working with distributed data” [7, 26]. Finally, the ARCADE (Architecture for Reusable Courseware Authoring and Delivery) Authoring Tool uses DTD and XSD content templates together with XSL presentation templates. As a result, it generates XML/WML course content and HTML course presentation [8]. This authoring tool supports hierarchies and could be extended by additional export modules. The ARCADE Authoring Tool is not immediately applicable to the context of RAGE, as the two projects have different goals and domains.

2.1. Design principals. The design of the metadata editor follows several core principles. The first one is *visual simplicity*. The editor hides the internal hierarchy and complexity of the metadata representation. An XML schema defines the metadata properties, relations and constraints. Some items, however, are implemented in different ways and the editor hides these differences. Thus it presents the metadata to the user in a unified way.

An example of such a difference in defining a metadata concept can be seen in the cardinality of attributes, which is defined in the schema as `use="required"`, and the cardinality of tags, which is defined as `minOccurs="1"`.

Another example is the use of controlled vocabularies, which are implemented in three ways:

- schema enumeration types `<enumeration>`;
- external taxonomies;
- `xml:lang` attributes.

Another core design principle is *flexibility*. It has significant impact on the internal realization of the metadata editor, which is not bound to a single fixed and predetermined metadata structure. Instead, it reads a schema about the metadata and reconstructs the structure of the metadata.

Currently the RAGE asset schema is composed of private RAGE-specific definitions enriched by industry-standard definitions from ADMS (Europeana's Asset Description Metadata Schema — a general schema for web-based assets [9]), DCAT (Data Catalog Vocabulary — provides interoperability between web-based data catalogues [10]), DCTERMS (Dublin Core Terms — a vocabulary describing resources in a searchable way [11]), FOAF (Friend-of-a-friend Schema — a semantic description of people and their social relations [12]) and RDF (Resource Description Framework — a specification for conceptual description of metadata models for web resources [13]).

The third underlying principle is abstraction. The RAGE Metadata editor is actually a meta-editor. It is not an editor by itself, but it builds a metadata editor in real time as pictured in Fig. 1.

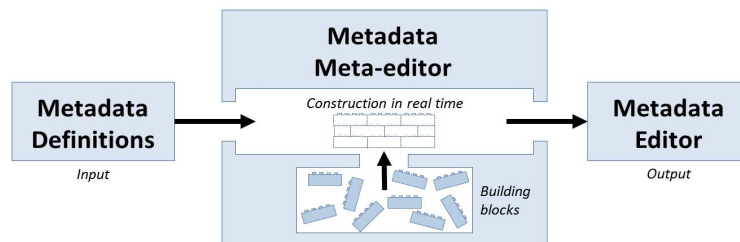


Fig. 1. Metadata meta-editor constructing a metadata editor in real time

The meta-editor is equipped with a collection of building blocks and construction algorithms. The input data contain heterogeneous definitions of metadata, like the metadata model itself, additional schemas, taxonomies and styling preferences. Then the meta-editor constructs the façade (the graphical user interface) of an editor, which is presented to the user.

The development of a metadata meta-editor, rather than merely a metadata editor, is a complex task. The reason for working at a more-abstract level is supported by the advantages that this level provides:

- Changes in the structure of metadata do not require changes in the meta-editor. This provides the flexibility to adapt the metadata model with minimal impact on other project software.
- If another metadata schema is given to the meta-editor, it will generate another editor. In this way the initial intent of building an editor of asset metadata is implemented as a general-purpose metadata editor that could process other RAGE entities like asset packages and artefacts.

These benefits come at a price. The most notable disadvantage of building a meta-editor is that it is a more complex piece of software and requires significantly more efforts. To ease future improvements and modifications of the meta-editor, it relies on widgets that encapsulate the underlying variety of metadata definitions.

2.2. User interface. A typical initial view of a RAGE Metadata Editor is shown in the left-hand snapshot in Fig. 2. The metadata elements are grouped into several tabs, called *Main*, *Classification*, *Status*, *License*, *Solution* and *Usage*. There is one additional tab, named *All*, which lists all metadata in a single and rather long web page.

Each tab (except for tab *All*) contains metadata elements in a specific category:

- *Main* — metadata about the general properties of the asset: title, description, type, date, language, and access URL.
- *Classification* — metadata describing and classifying the asset: keywords and taxonomy classifications.
- *Status* — metadata about the current version and development status, as well as the asset's relation to other assets: version, version notes, status,

maturity level, custom metadata, related assets and dependencies with other assets.

- License — metadata about the people and organizations related to the asset and its license: creator, publisher, owner, and license.
- Solution — metadata about the software and components that build up the asset: description, requirements, implementation, design, engine, platform, programming language, and tests.
- Usage — metadata about asset installation, customization, configuration and usage.

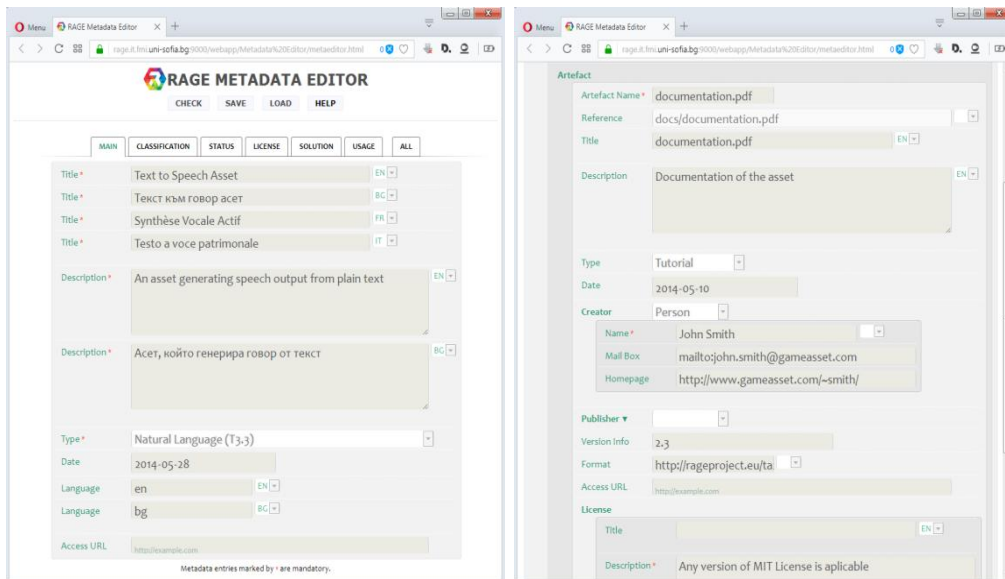


Fig. 2. The opening page (left) and the metadata hierarchy (right) in the RAGE Editor

The last two categories contain metadata for artefacts (these are asset resources, usually data or documentation files), which have their own set of metadata, such as name, reference, type, creator, version, format, license, etc.

The metadata elements are positioned vertically from top to bottom and follow the order and the nesting defined in the metadata schema. The actual appearance of the elements depends on the browser. The hierarchy of the metadata model is preserved in the editor. Nested metadata are

represented as nested blocks (see the right-hand snapshot in Fig. 2), which can be expanded, collapsed, duplicated or deleted.

Most of the metadata use standard user interface control elements like text boxes, list boxes and buttons. The only exception is the section for asset classification, where users select concepts from taxonomies, relevant to the asset. The visualization of the classification section is managed by another RAGE tool — the Taxonomy Selector.

The role of the meta-editor is to embed this tool in the front page and to manage the data transfer between itself and the tool. A snapshot of the selector is shown in Fig. 3 with a fragment of the hierarchy of the ACM Computing Classification System of Applied computing.

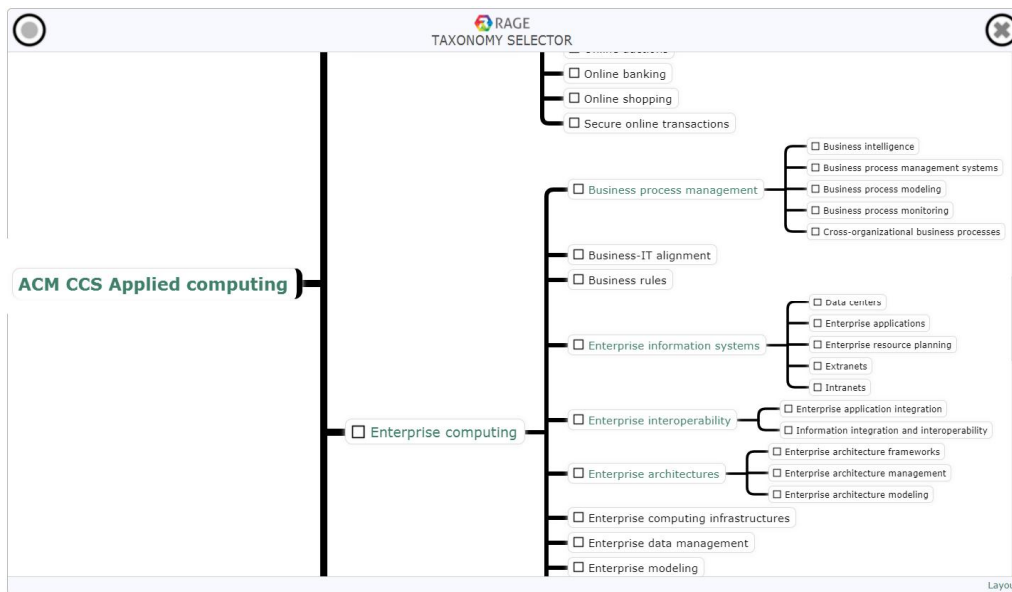


Fig. 3. The RAGE Taxonomy Selector

2.3. Metadata verification. The definition of assets' metadata in RAGE can happen through several data streams. The metadata meta-editor is one of these streams. Other ways to get metadata are via direct upload of assets or via harvesting. Thus the verification module, responsible for verification of the metadata of each incoming or edited asset, is not a component of the meta-editor. As a result, the meta-editor does not provide a

full verification, but it executes some rudimentary measures to partially verify the metadata entered by the user. Some verification items are checked explicitly, others are embedded in the user interface layout. Details are provided in Table 1.

Table 1. Partial verification within the meta-editor

Item	Description	Means of verification
Metadata structure	Asset metadata is a complex hierarchical structure conforming to the metadata model.	The metadata model is encoded in the metadata schema. The meta-editor creates the user interface in accordance with the schema. The location of metadata in the model's hierarchy is enforced on the user interface.
Missing metadata	Some metadata are optional and may be omitted, other are compulsory.	There is a visual indication for compulsory metadata — red asterisks beside the labels. Additionally, when the metadata are prepared for the server, the user is notified about all missing compulsory metadata.
Conditionally missing metadata	Some compulsory metadata are inside an optional block. They are treated as compulsory only if the block is not otherwise empty.	When the metadata are prepared for the server, the editor explicitly checks all compulsory metadata inside optional metadata. This check is performed on the full depth of the hierarchy. The user is notified if there are missing compulsory data in optional blocks.
Date metadata	Some metadata hold dates as strings.	The editor uses the HTML5 date input type. If the browser supports it, then the user will fill a predefined template for the date or select a date interactively. For older browsers the date metadata is accepted as a string without verification of the contents.
Schema-based controlled vocabulary	Some metadata may have a value from a fixed list of values coded in the schema.	The editor generates a list box with the available values, so the user cannot select a value which is not allowed. The only exception is the “empty” value, which is identical for lack of data.

Taxonomy-based controlled vocabulary	Some metadata may have a value from the RAGE taxonomy.	The editor generates a list box with the available taxonomy concepts, so the user cannot select an unknown concept. The only exception is the “empty” value, which is identical for lack of data. If taxonomy is changed and some metadata become invalid, they will be skipped by the editor as if they were “empty” values.
Classification taxonomy	The user may add a classification to an asset based on selected concepts from selected taxonomies.	The editor allows the user to select only taxonomies which are compatible with the taxonomy tools and exist in the RAGE repository. When the user selects concepts from these taxonomies, the same verification as in the taxonomy-based controlled vocabulary is used.

Apart from this partial verification, the meta-editor has a special developer’s mode. This mode is used during the development of the meta-editor while the actual asset metadata records are still incomplete or incorrect. When the tool works in this mode, it provides additional internal details like links between visual elements and the metadata schema, log files of metadata properties, toggling metadata visibility, etc.

In addition to the developer’s mode, the meta-editor may be configured at run-time. It may change the source of metadata (retrieving metadata from the server or from a local file), the visual appearance (by using alternative CSS files), or the metadata schema.

3. The User-centric approach. The second approach to designing and implementing a metadata editor is focused on the user and the user experience. This shifts both the design and the internal architecture of the editor. While the data-centric meta-editor is targeted towards experienced users, who know in detail the metadata model of RAGE assets, the user-centric metadata editor is targeted to external asset developers, who are not expected to be familiar with how the description of assets is presented in the metadata records. As a result, the user-centric metadata editor is closer to a wizard software, rather than to a general editor. Therefore this metadata editor is named *RAGE Asset Metadata Wizard*.

Although both the meta-editor and the wizard are used to modify the same record of metadata, they follow completely different approaches. This significantly affects the implementation of both tools. The metadata wizard is built from scratch. It does not use any internal components from the meta-editor. The data processing is straightforward, because the interface and the analysis of metadata are hardcoded in the source code. The wizard provides several advantages over the data-centric meta-editor:

1. The wizard guides the asset developer through several steps of describing the asset.
2. The interface is simple, easy to use and self-explanatory.
3. The effective use of the wizard requires no preliminary knowledge of the metadata model.
4. There are indicators of how complete and accurate the asset description is.
5. Verification is done as early and completely as possible.

The following subsections discuss some details of several of these properties of the asset metadata wizard.

3.1. User interface. Considering the actual needs of the contents of the asset description, the wizard shows only a minimal set of 20 compulsory and 14 optional preselected metadata elements. The other elements are not used by the wizard, but they are still accessible by the data-centric meta-editor. The preselected metadata elements are semantically clustered into eight groups. Each group forms a step of the asset description process — see table 2.

Table 2. Semantic grouping of metadata

Group / page	Description	Metadata elements
About	General information about the asset, e.g., title, description, logo, access URL, etc.	5 compulsory 3 optional
Classification	Information about target platforms, programming language and applied computing keywords.	1 compulsory 5 optional
Status	Contains software version, version notes, commit reference, development status.	3 compulsory 1 optional

License	Details about licenses, conditions and potential restrictions.	2 compulsory 1 optional
Contacts	Information about owners and creators of the asset.	2 compulsory
Resources	Files or references of the software, documentation, tests, etc.	3 compulsory 3 optional
Quality	Information about the asset's quality and self-declaration form.	4 compulsory 1 optional
Submission	Review of asset description completeness; asset submission.	No metadata elements

The appearance of the wizard is presented in Fig. 4. The snapshot on the left is the first step containing the general asset description (group About); the snapshot on the right is the 7th step, which contains the self-declaration form (group Quality).

Software Asset Wizard

Step 1 About 90% completed

Welcome to the RAGE Software Asset Wizard. It will guide you through the process of describing your asset. First we'll create a general description of the asset.

Step 2 Classification

Step 3 Status

Step 4 License

Step 5 Contacts

Step 6 Resources

Step 7 Quality

Step 8 Submission

Name
This is the first information that a user will see about your asset. Try to think of a name that is informative, short and engaging.

Performance Statistics Asset

One sentence description
Please provide a one-line description of what your software asset can do. This text is crucial as it will show up in the search results and helps users to decide whether or not they want to continue with the software asset.

This asset automatically performs statistical analyses on players in-game performance and makes a comparison to

Short non-technical description
Please explain in 3-4 lines without technical details what your asset can do. This text will be used for short advertisements on the web and in search results.

This asset is linked through a game and it produces a visual representation of player and group performance statistics. The asset also provides warnings if users should be careful in interpreting these statistics.

Technical description
Please briefly explain some of the technical details of the asset. Explain: (1) what the asset does; (2) what it could be used for; (3) what inputs are needed; (4) what outputs it produces.

This asset is a custom analytics solution in the UCM server. It receives game data through the UCM tracker and produces a dashboard containing a visual overview of the performance and related warnings and errors in statistical

Software Asset Wizard

Step 1 About 80% completed

Step 2 Classification

Step 3 Status

Step 4 License

Step 5 Contacts

Step 6 Resources

Step 7 Quality

Step 8 Submission

Software Quality
Finally, we need information about the software quality of the software asset and a declaration about the accuracy of your software asset description.

Coding style
Considering the software components of the software asset, please, select an option that describes the coding style.

Individual programmer style

Architecture
If your software asset is client-side, indicate if it is compliant with the RAGE client architecture. Please refer to architecture requirements for C# or TypeScript.

The software asset is server-side

Software testing [optional]
Please, select all successfully performed tests of the software asset.

Unit tests
 Integration tests with RAGE's Software Asset Manager
 Integration tests within a game platform
 End-to-end tests
 Performance tests

I declare that:

The information that I have entered is correct to the best of my knowledge
 I take sole responsibility for all content posted and activity that occurs under my account

Fig. 4. Snapshots from the RAGE Software Asset Wizard

All pages share the same visual style. The left-hand part contains the sequence of steps (1 to 8), the top contain an explanation about the step. Each metadata element in these pages has its own title and description, which are used to reduce the user confusion about the purpose of the metadata.

3.2. Metadata verification. The top right corner of each page of the wizard shows the percentage of completeness of the metadata on the page. Showing an indicator of completeness is a major requirement for the wizard, along with the requirement of guiding the user through a series of steps.

While asset developers describe their software assets in the wizard, it continuously calculates and updates two scores. Apart from the completeness, there is also a score for the asset quality. Both scores are recorded in the metadata of the asset, but users cannot access and modify them directly.

The scoring algorithm of the wizard is designed to provide as much detail and granulated verification as reasonably possible. Counting just the presence of metadata and considering whether it is compulsory or optional does not provide enough details — this would be just a rough indicator for completeness.

Although the metadata completeness is shown in the top right corner of each wizard’s page (see snapshots in Fig. 4), they are also summarized in the last page of the wizard. Fig. 5 is a snapshot of a fragment of this page.

Step 1 About	Submission 83% overall completed	
Step 2 Classification	This is the last step. Please, take a moment to review the completeness of your asset description. If necessary, you can go back to any of the previous steps and change or update your description.	
Step 3 Status	1. About	90%
Step 4 License	2. Classification	57%
Step 5 Contacts	3. Status	81%
Step 6 Resources	4. License	100%
Step 7 Quality	5. Contacts	98%
Step 8 Submission	6. Resources	75%
	7. Quality	80%
	OVERALL:	83%

Fig. 5. The last step of the wizard with metadata completeness scores

The user describing an asset can see the completeness for each group of metadata elements, as well as the asset’s overall completeness. If compulsory elements are missing, then the wizard shows a warning.

The wizard fine-tunes the scoring by attaching verification rules for each metadata element. Most of the rules inspect not only the presence or absence of metadata, but also the contents of the metadata. Thus the wizard uses an approach which could capture minor differences based on the actual values. The list of defined rules is presented in Table 3.

Table 3. Verification rules in the wizard

No	Rule	Description
1	Default value	The metadata value is the same as the default value provided automatically by the wizard.
2	“Others” value	The “others” option from a list of predefined values is selected, so the actual value is to be provided as custom metadata.
3	Common value	The value is too common (e.g., the name of the asset is “RAGE asset”); a more specific value is expected.
4	Empty	The value is missing or is an empty string.
5	Optional empty	The value of an optional metadata element is missing or is an empty string.
6	Internal link	The value is a link, URL or URI to a local resource or service within the RAGE server.
7	Length (N)	The number of characters in the value is less than N.
8	Words (N)	The number of words in the value is less than N.
9	Phrases (N)	The number of comma-separated phrases is less than N.
10	Sentences (N)	The number of sentences in the value is less than N.
11	No connection	No connection to the RAGE server, the value is checked against cached taxonomies.
12	Version	The version number not N.N, N.N.N or N.N.N.N.
13	Invalid date	Either the date or its formatting is invalid.
14	Strange date	The date is too old or it is in the future.
15	Custom metadata (N)	The formatting of the custom metadata is not split into N or more pairs name=value.
16	URL	The URL is not correct.
17	Path	The absolute or relative path is not correct.
18	Email	The email address is not correct.
19	English	The value does not appear to be a text in English.
20	Always	This rule is always applicable — it is the termination rule.

Each metadata element has a maximal weight from 0 to 10, defining its importance. The most important elements, like the asset title, have a weight of 10. Less important ones have smaller weights, e.g. the development status is 8, the programming language is 6, the asset date is 5. Depending on the actual value, some metadata may have their weights reduced. This reduction contributes to the fine granulation of completeness indicators.

The overall completeness score is the ratio of the total actual weight of all metadata (from 0 to 236) and the total of their maximal weight. Each of the steps in the wizard has its own completeness score, which is calculated in the same way as the overall score, but it considers only the metadata from the step. Table 4 shows the maximal score of each group and its contribution to the overall score. The distribution of weights reflects the relative importance of the metadata groups, thus the two most important fragments of the metadata are the general descriptions of the asset and its resources.

Table 4. Weight of metadata groups.

Group	Maximal Score	Contribution to the total score	Number of rules
About	62	26.3%	32
Classification	28	11.9%	8
Status	31	13.1%	11
License	25	10.6%	6
Contacts	25	10.6%	12
Resources	40	16.9%	10
Quality	25	10.6%	4
Submission	-		7
Total	236	100%	90

There is a list of rules attached to each metadata element (the total number of rules in each metadata group is shown in the last column of Table 4). Each rule verifies a specific aspect of the metadata value and reduces the weight of the element if the rule is not followed.

The score of a group of metadata elements M_i and the score of the whole asset metadata is calculated as:

$$S = \sum \left(1 - P_{i, \min(j):A_{i,j}}\right) S_i$$

where S_i is the maximal score of M_i , $P_{i,j}$ is the reduction factor (penalty) from the j -th rule for M_i ; and $A_{i,j}$ is *true* if the j -th rule is applicable or effective to metadata element M_i .

When the metadata value is checked, only the first failing reduction rule is applied — this is the reason to use $\min(j)$ in the formula. For example, the metadata element containing the asset's keywords has a maximal initial weight of 3. The value is evaluated by checking its rules from top to bottom, stopping at the first applicable rule. The rules (see Table 3) for the asset's keywords are:

- **Rule 4:** If the value is empty, then the element's score is 0 (i. e., 100% reduction)
- **Rule 15:** If the value contains less than 3 keywords, then the score is 2.7 (i. e., 10% reduction)
- **Rule 19:** If the value looks like non-English text, then the score is 1.5 (i. e., 50% reduction)
- **Rule 20:** If none of the above rules are activated, then the score is 3 (i. e., no reduction at all).

Some metadata elements have only one or two rules, while other may have up to six rules. Usually, the more complex or important element is, the more sensitive it is to its content, and more rules are checked.

An example of such a sensitive element is the asset description. Its rules are: rule 4 (the description is empty), rule 1 (the text is same as the default value), rule 7 (is it less than 10 characters), rule 8 (is it less than 3 words), rule 10 (is it just one sentence); rule 19 (does it sound like a text in English) and rule 20 (termination rule). Because the rules are embedded in the code of the wizard, the source of the verification of the asset description looks like this:

```

setPenalty(id, 0);
if (stringEmpty(str))      return setPenalty(id, 1.0);
if (stringDefault(str, 'Empty RAGE Asset description.'))
    return setPenalty(id, 0.5);
if (stringLength(str, 30)) return setPenalty(id, 0.2);
if (stringWords(str, 10)) return setPenalty(id, 0.1);
if (stringSentence(str))  return setPenalty(id, 0.1);
if (stringEnglish(str))   return setPenalty(id, 0.5);

```

where `id` is the metadata element, `str` is its value as a string, `setPenalty` sets the reduction factor and the family of all `stringXXX` functions are the software implementations of the rules. This code fragment also shows that the reduction factors for the rules are not fixed, because the same rule for different metadata elements may impose different impact.

Many of the rules are general checks, like number of words, out of range dates, etc. Other rules are based on regular expressions, like rule 18 that verifies syntactically an e-mail address `/\S+@\S+\.\S+/\` or rule 16 for checking a resource URL `(http|ftp|https)://[\w-]+(\.[\w-]+)+([\w-.,@?^=%&:/~+#-]*[\w@?^=%&:/~+#-])?`.

The most complex implementation is that of Rule 19, which checks whether some text sounds like English. It is neither feasible to have a huge dictionary with English words, nor to access external linguistic services. Instead, the wizard adopts its own statistical algorithm, which evaluates the “Englishability” of some text by evaluating three factors: variety of character pairs V_P , frequency of character pairs F_P and variety of characters V_C ,

$$E = 10V_P + F_P - V_C = 10 \frac{|pairs|}{|text|} + \frac{\sum(F_{i,j} - 15)}{|text|} - \frac{\min(|text|, 26)}{|chars|}$$

where $|text|$ is the length of the text in characters, $|chars|$ is the number of distinct characters, $|pairs|$ is the number of distinct pairs of characters and $F_{i,j}$ is the frequency of specific pair of characters based on seven novels [14].

The calculated value E is a metric for the “Englishability” of the text. If $E > limit$ for some limit, then the text is considered to be in English.

The formula is shaped experimentally by testing with different metadata strings, harvested from existing asset descriptions. The most influential and English-language-specific factor is the frequency component F_P . The other two factors are present to fine-tune the result, because the frequency statistics for short texts does not provide adequate results.

For most elements the limit for E is set to 20, except for the keywords, which have a limit of 14, because they may contain non-English text like acronyms, abbreviations, product version numbers, etc. Table 5 shows the calculated Englishability of two English texts and three texts which are not judged as English.

Table 5. Englishability of some texts

Text	Englishability				English?
	$10V_P$	F_P	V_C	E	
Empty RAGE Asset	6.88	19.44	1.78	24.53	Yes
This asset is linked through a game and it produces a visual representation of player and group performance statistics.	5.04	40.76	1.30	44.51	Yes
Asdf asdf asdfasdfsdf	1.82	16.36	5.50	12.68	No
Blah-blah	3.33	3.78	2.25	4.86	No
9c8w37nroc iuerh ncs8qe	4.35	5.91	2.09	8.17	No

3.3. New metadata elements. The RAGE Metadata wizard is developed after the metadata editor. This changed some of the requirements for the wizard with respect to the metadata. Namely, several new metadata elements were suggested for inclusion in the wizard such as several types of descriptions (short, long, general, technical), asset logo, asset artefacts, quality assurance self-declaration form, etc.

Some of these elements were not present in the metadata model. Therefore, if the data-centric meta-editor were modified to support these elements, that would require a change in the metadata model itself. The wizard, however, has its own metadata model, which is transparently translated to and from the official RAGE metadata model. The difference between the two models reflects the difference between what users see when they use the data-centric and the user-centric editors.

The metadata elements that are dropped out of the wizard are easy to handle. Optional elements are just ignored, while compulsory elements are filled in with predefined stock values.

The new metadata elements are the actual challenge for the wizard, because one of the initial requirements is that the wizard uses the same official metadata model as the meta-editor. After collecting the revised requirements for the wizard, it was possible to summarize the new metadata elements:

- *Asset quality* — the value is a numeric score, calculated upon a self-declaration form.

- *Detailed description* — a new longer description of the asset.
- *Promotional description* — another variant of the description, which is used in promotional materials.
- *Technical description* — description with technical details about the platform, the software, the protocols, etc.
- *Commit URL* — a link to GitHub commit resource for the asset, useful for the asset developers and asset users.
- *Asset completeness* — the automatically calculated completeness score is stored in the metadata, so that other tools may access it directly.
- *Coding style* — a part of the self-declaration form, describing the coding style of the asset software.
- *Architectural conformance* — describes the asset conformance to a set of requirements and specification.
- *Software testing* — lists the types of internal tests passed successfully by the asset.
- *Self-declaration* — indicates whether the self-declaration form is completed.

The solution adopted in the wizard is to implement new metadata elements as *custom metadata* elements. The official metadata model contains custom, user-definable elements with the sole purpose of extending the model with new metadata.

To make a seamless transition between the two metadata models, the wizard automatically converts custom metadata from the official metadata model to normal metadata elements in the wizard's model and vice versa.

The only way for a user to understand that some metadata are stored as custom metadata is to view the asset through the data-centric meta-editor, because it shows the metadata as it is stored. The other option is to monitor and inspect the actual XML data stream between the wizard and the server.

Except for these new metadata elements, there are other elements which the wizard supports — these are resources (artefacts) such as data files, source code, tutorials, documentation, configuration files, etc. Although these resources are included in the RAGE metadata model, they are not implemented in the meta-editor in the same way as in the wizard.

The incoming metadata have two indicators of each resource — its type and its location section within the metadata model; however, sometimes these two indicators conflict with each other. On the other hand, the wizard supports only sections, but not the same set of sections as the metadata model. To resolve this discrepancy, the wizard translates incoming types and sections into wizard's sections. Before the asset metadata are sent to the server, the wizard's sections are converted back into metadata sections. The conversion mapping of resource types is shown in Fig. 6 — the incoming metadata is on the left, the resulting outgoing metadata is on the right.

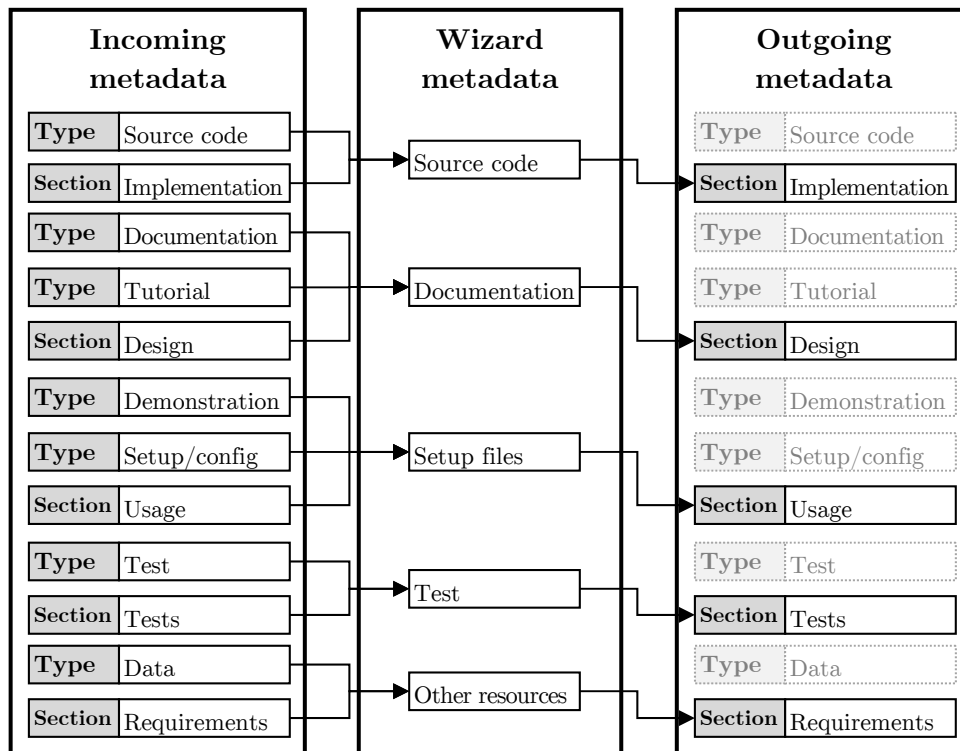


Fig. 6. Conversion of resource types

Finally, there is yet another new metadata element, which is also a resource — the asset logo — an image, shown along with the asset title and description both in the RAGE portal and in promotional materials. The wizard handles the asset logo as a standalone metadata element. Internally, it is represented as an image resource and is stored as one of the asset's artefacts.

3.4. Web portal. When users access the metadata of software assets developed by others, they are not allowed to modify them. The data-centric meta-editor uses a single interface — when data modifications are forbidden, the *Save* button is removed from the web form and all data entry elements are set to read-only mode and are greyed out to indicate visually their mode.

The wizard assists the describing of the asset’s metadata, but the interface is not tailored for the case when users just want to view the asset metadata. To address the problem another tool is developed — the metadata viewer.


Client Side Game Storage Asset

Download

All assets

Your current profile can not edit this software asset

About

Name: Client Side Game Storage Asset 

One sentence description: Provides easy storage of and access to treelike data structures.

Short non-technical description: Provides storage of multiple treelike data structures. Data can be stored either locally or remotely (using the Game Storage Server Asset). Values can be accessed by name. Structure and data are saved separately. Each node can specify a preferred storage location.

Technical description: Provides storage of multiple treelike data structures without making assumptions on what data is actually stored. The structure is saved independently of the data. Each node can specify a preferred save location and the asset is able to use this information to save data in multiple location like on-device using at the game storage server.

Language: English

Access URL:
<https://github.com/rageappliedgame/ClientSideGameStorageAsset>

Classification

Game development environment: Other

Target platform: Other

Programming language: C#

Applied computing concepts: Education

Keywords: data-storage, c#, mobile

Status

Version: 0.9.0

Version notes: Initial version.

Development status: Under development

Commit URL:
<https://github.com/rageappliedgame/ClientSideGameStorageAsset/commit/125476f61c1383cce4f19b2d5dcdea922d63ee5d>

Fig. 7. The RAGE Metadata Viewer

The viewer extracts the metadata of a software asset and arranges them in a structured page as shown in Fig. 7. This interface is suitable for a quick overview of an asset, because its metadata are presented in a more

compact style; usually they fit in a page or two. Additionally, the interface is printer-friendly and the asset description could be printed as a hard copy of the asset dossier.

The metadata viewer shows only the metadata which are processed by the wizard. To see the full metadata, the user must use the meta-editor.

Although the purpose of the viewer is to show the metadata, it is also the intermediate layer between the front-end asset manager and the asset wizard — see Fig. 8. Asset users browse and search all assets in the repository within the asset manager. When they click on a selected asset, it is opened in the asset viewer. Then the users can inspect the asset description and download it if they wish to incorporate it in their game. However, if the users are the developers of this asset or if they have sufficient write permissions, they can further open it in the wizard to edit it. When finished editing the asset, they automatically return to the viewer to review the changes.

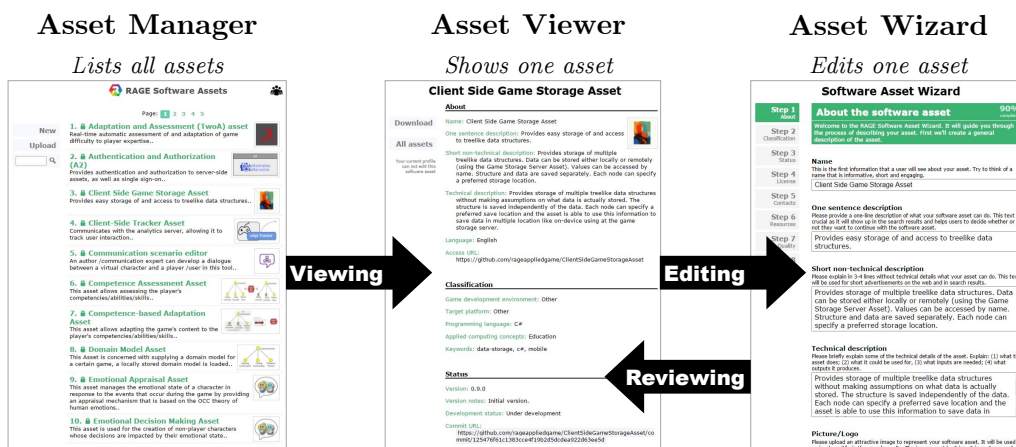


Fig. 8. The viewer as an intermediate layer between the asset manager and the wizard

The first two tools – the asset manager and the asset viewer – are available to all RAGE users. The third tool, the asset wizard, is only available to asset developers.

4. Comparison of the two approaches. The first working prototype of the meta-editor was developed in early 2016. Its purpose is to edit

the asset metadata by exposing the full complexity of the metadata model. An internal review in 2017 revealed that asset developers outside the RAGE consortium might find it difficult to use the data-centric meta-editor, as they are not familiar with the metadata mode structure, nor are they expected to.

After a short but intensive design phase, the RAGE Metadata Wizard was developed in March and April 2017. Although it has the same purpose — to edit asset metadata — its approach and implementation are conceptually different. Instead of being data-centric like the meta-editor, the wizard is designed as a user-centric tool. It is not possible to compare the two tools in terms of better or worse, because they both have their own specific advantages and disadvantages.

For low-level work (internal asset developers and RAGE developers) the data-centric meta-editor might be more appropriate, because it shows the metadata as they are — both their contents and their hierarchy. For high-level work (external asset developers and RAGE users) the user-centric wizard is more suitable, because it hides the complexity of the metadata and presents them in a comprehensive way. A side-by-side comparison of the meta-editor and the wizard is presented in the following table 5.

Table 5. Comparison of data-centric and user-centric metadata editors

Criteria	Data-centric Meta-editor	User-centric wizard
Developer's perspective		
Implementation efforts	High	Moderate
Development time	An year	A month
Supporting and upgrading effort	No effort for changes in the metadata model Significant effort for changes beyond that	Moderate effort
Performance	Slower, data pass several transitions	Faster, processing is computationally simple
User interface	Generated in real-time	Prebuilt and fixed
Metadata verification	Rudimentary	Complete, except for artefacts

User's perspective		
Target users	Asset developers and data administrators	External asset developers
Complexity of metadata presentation	Hierarchical	Flat and simplified
Filling metadata elements	Homogeneous, unguided	Heterogeneous, guided
Indicators	Only missing compulsory metadata	Detailed indicators of metadata completeness
Graphical interface	Focus on completeness and accuracy	Focus on navigation and comprehension
Viewing others' assets (read-only mode)	Same interface. All UI elements are forced into read-only mode	Separate tool — the metadata viewer

In terms of implementation, the data-centric meta-editor is much more complex, with its cascading data flow through several metadata-processing phases. The development took about a year, but the flexibility in the design allows upgrading and improvement — the meta-editor generates an editor in real time following a metadata schema. Therefore, if the metadata are modified, the meta-editor will create another editor. However, if the changes go beyond the schema, then the effort to implement them would be significant.

The wizard has a moderate complexity; its implementation took a month because it is relatively simpler and straightforward. As all metadata elements are hardcoded in the wizard, any modification, even the slightest one, would require modification of the source code.

Because of its internal complexity, the meta-editor has slightly lower performance; the user interface needs a second to be generated, because several schema files and taxonomies are downloaded over the internet. The tool performs only rudimentary metadata verification, relying on the server to run a complete verification.

The wizard, on the other hand, is faster, its interface is prebuilt, and it performs complete verification of the metadata in real-time. The only exceptions are the artefacts — the wizard expects the RAGE server to signal back problems with the uploaded resources, like duplicate names, unsupported file types, etc.

In terms of users' experience, the meta-editor and the wizard are also quite different. The target users of the meta-editor are internal asset developers and data administrators, who are familiar with the metadata model. These users can work with the full complexity of the metadata hierarchy, including several levels of nested metadata blocks. The wizard is more appropriate to external asset developers, who are not expected to know the details of the metadata model. This is because the wizard shows a flat oversimplified list of elements.

In terms of the user interface, the RAGE meta-editor indicates only missing compulsory data, while the wizard shows detailed indicators of metadata completeness, accompanied by hints and suggestions. Additionally, the wizard's interface is focused on easier navigation and guided data entry.

5. Conclusion. This paper presents the main aspects of two metadata editors, designed and developed for the Horizon 2020 project RAGE. Both editors are used to edit the metadata of RAGE software assets — these metadata contain descriptions and additional details about the assets and are used to search and classify them.

The first metadata editor is based on the data-centric approach. It provides a complete unabridged view of the metadata of an asset. This exposes the full metadata hierarchy and internal structure. This editor is implemented as a meta-editor — i. e., it reads the metadata descriptions of an asset and builds a corresponding metadata editor in real time. The user interface is with nested blocks that recreate the metadata hierarchy. The other editor follows the user-centric approach. It reshapes the metadata model and presents it to the users in a more manageable way. Additionally, the wizard guides the user through the process of describing an asset, while providing instant feedback of the description's completeness and accuracy.

To utilize the full functionality of both editors, they need additional tools. The meta-editor uses a taxonomy selector, embedded in the generated editor, while the wizard is connected to the portal via a metadata viewer.

Although both metadata editors have the same purpose – to describe an asset via metadata – their designs, internal architectures, implementations and appearances are completely different. This difference significantly affects

the user's experience. As a result, the data-centric meta-editor is more appropriate for experienced users, who need to view and access the metadata in their full complexity; while the user-centric wizard is more suitable for the general asset developers, who are focused on the development of their assets and do not need to be familiar with the RAGE metadata model.

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] What is Horizon 2020? European Commission, The EU Framework Programme for Research and Innovation. <http://ec.europa.eu/programmes/horizon2020/en/what-horizon-2020>, 15 October 2018.
- [2] RAGE. Project website. <http://www.rageproject.eu>, 15 October 2018.
- [3] VAN DER VEGT W., W. WESTERA, E. NYAMSUREN, A. GEORGIEV, I. ORTIZ. RAGE Architecture for Reusable Serious Gaming Technology Components. *International Journal of Computer Games Technology*, 2016, Article 3. DOI: 10.1155/2016/5680526
- [4] GEORGIEV A., A. GRIGOROV, B. BONTCHEV, P. BOYTCHEV, K. STEFANOV, K. BAHREINI, E. NYAMSUREN, W. VAN DER VEGT, W. WESTERA, R. PRADA, P. HOLLINS, P. MORENO. The RAGE Software Asset Model and Metadata Model. In: JCGS 2016: Serious Games. *LNCS*, **9894** (2016), 191–203.
- [5] SULEMAN H. Metadata Editing by Schema. In: T. Koch, I. T. Sølvsberg (eds). Research and Advanced Technology for Digital Libraries, 7th European Conference, ECDL 2003, Trondheim, Norway, 2003. *LNCS*, **2769** (2003), 82–87.
- [6] GRASSO M., M. CRAGLIA. D 2.2.3A: European Open Source Metadata Editor Developers' Guide v.1.0. EuroGEOSS, 2010. http://www.eurogeoss-fp7-project.eu/Documents/EuroGEOSS_D_2_2_3A.pdf, 15 October 2018.

- [7] KUNZE T., J. BRASE, W. NEJDL. Editing learning object metadata: Schema driven input of RDF metadata with the OLR3-Editor. In: Semantic Authoring, Annotation & Knowledge Markup Workshop, 2002, 22–26.
- [8] BONTCHEV B., T. ILIEV. ARCADE — Web-based Authoring and Delivery Platform for Distance Education. In: First Balkan Conference on Informatics, Thessaloniki, Greece, 2003, 293–306.
- [9] DEKKERS M. Asset Description Metadata Schema (ADMS). W3C Working Group Note, 2013. <https://www.w3.org/TR/vocab-adms/>, 15 October 2018.
- [10] MAALI F., J. ERICKSON (eds). Data Catalog Vocabulary (DCAT). W3C Recommendation, 2014. <http://www.w3.org/TR/vocab-dcat/>, 15 October 2018.
- [11] DCMI Metadata Terms. Dublin Core Metadata Initiative, 2012. <http://dublincore.org/documents/dcmi-terms/>, 15 October 2018.
- [12] BRICKLEY D., L. MILLER. FOAF Vocabulary Specification 0.99. 2014. <http://xmlns.com/foaf/spec/>, 15 October 2018.
- [13] SCHREIBER G., Y. RAYMOND (eds). RDF 1.1 Primer. W3C Working Group Note, 2014. <http://www.w3.org/TR/rdf11-primer>, 15 October 2018.
- [14] LEON J. S. Frequency of Character Pairs in English Language Text. Codes and Cryptography, 2008. http://homepages.math.uic.edu/~leon/mcs425-s08/handouts/char_freq2.pdf, 15 October 2018.

Pavel Boytchev

*Department of Information Technology
Faculty of Mathematics and Informatics
St. Kliment Ohridski University of Sofia
5, James Bourchier Blvd
1164 Sofia, Bulgaria
e-mail: boytchev@fmi.uni-sofia.bg*

Received October 20, 2017

Final Accepted June 20, 2018