Interacting with Semantic Data by Using X3S

Timo Stegemann, Tim Hussein, Werner Gaulke, and Jürgen Ziegler

University of Duisburg-Essen
Lotharstr. 65, 47057 Duisburg
firstname.lastname@uni-due.de

Abstract. The Internet has transformed increasingly from a document-centered web to a data-oriented one, enabling new opportunities to both users and developers. However, interacting with this data in a traditional way is often limited due to its amount and heterogeneity. In this paper, we describe X3S, an approach to filter and illustrate semantic content of data-webs, allowing the creation and use of components for data-exploration and visualization. In addition to that, we introduce an editor for X3S files, facilitating the creation of and the interaction with semantic data. The combination of the X3S format and the editor is compared to existing approaches and evaluated in a user study.

1 Introduction

Based on techniques and initiatives such as RDF and Linking Open Data, respectively, the available amount of semantic data steadily increases. Among others, electronic commerce intensely utilizes semantic representation of product data [4] and offers [5]. On the Internet, a shift from a rather document-centered web to a more data-oriented one can be observed as well. However, handling extensive, interconnected data may be cumbersome and not intuitive, which is why common methods of web engineering often reach their limits in this respect.

In this paper, we propose a method and a format for interacting with semantic data by defining templates following the X3S specification. X3S is an acronym for "XSL-transformed SPARQL results and Semantic Stylesheets". In contrast to CSS, known from web design, X3S includes directives for filtering data (by using SPARQL) as well as for decorating it (by using XSL transformations and CSS). In order to create and maintain X3S files, we developed an editor enabling the user to create complex templates for querying and presenting semantic data intuitively. Both the basics of the X3S format as well as the editor are introduced in this article.

2 Related Work

Several RDF or semantic web browsers have been developed to explore semantic data sources. However, presentation often is limited to tabular listings of all instances, properties, and property values. Examples are *Tabulator*[1] and *Disco*[2]. They work well for

[1] http:www.w3.org/2005/ajar/tab
[2] http://www.wiwiss.fu-berlin.de/bizer/ng4j/disco

getting a simple overview of data sets, yet they don't support styling or filtering of instances. As X3S is a solution with rich interaction for semantic data, we look at related projects with a similar focus.

The *Xenon* project [8] introduced an ontology described as "stylesheet ontology" or "RDF stylesheet lanuguage". The goal of *Xenon* is to display semantic data in a human usable way and facilitate the alteration of the representation according to the user's needs. Based on the concept of XML-Stylsheets (XSLT), the authors created a RDF-based stylesheet, which defines concepts for transforming RDF-data. The concept includes *lenses* and *views*. The purpose of lenses is data selection from instances whereas views describe the visualization of data elements. Just like with XSLT it is possible to embed HTML markup directly into the stylesheet for generating HTML representations of the data.

The RDF vocabulary *Fresnel* was developed as a successor of *Xenon*. Fresnel still uses lenses to select data but dropped the views concept and with it the possibility to embed visual markup such as HTML [7]. As a replacement, the *formats* concept is introduced. They are used for formatting data and enrich it with additional information for the renderer. For example, a format defines whether the selected lens data should be handled as a link, an image, or as a text. This added information is used by the browser for creating the final visualization. For example, the data could be represented as a table or graph, depending on the decision of the browser. To select data, lenses can use a special Fresnel Selector Language (FSL) or standardized SPARQL. Furthermore, lenses can be used in cascades for breaking data selection into several smaller pieces, thus improving reusability.

With *OWL-PL* [3], a language for transforming RDF/OWL data into (X)HTML is introduced. The language is strongly inspired by XLST and has the main goal to provide a simple transformation language for semantic data. *OWL-PL* allows the combination of transformational and representational markup. The language defines stylesheets, which are connected to semantic data by using a stylesheet ontology. The ontology describes how specific RDF classes are related to stylesheet elements. For example, the most universal relation for OWL:thing could be related to a tabular representation. If several relations are given for a class, the user can decide which graphical representation should be used. With help of a server-side Java application, stylesheets are converted into HTML.

With the introduction of *LESS* [1], a complete workflow from creating and processing templates for semantic data up to sharing templates between users is described. The declarative template language LeTL (LESS template language) is specified as a Smarty based templating language. It can process and transform semantic data from RDF documents or data requested by SPARQL queries. LESS provides an editor for creating LeTL templates. The editor shows available properties from the RDF data used or the SPARQL result. The properties can be directly used in LeTL code and combined with HTML markup to generate the final HTML output (Figure 1). The editor is not tied to HTML, other output formats can be created as well. The created LESS template is saved on the LESS server and can be accessed through a REST-based interface.

Contrary to the other approaches introduced in this section, *Dido* (Data-Interactive DOcument) [6], does not process semantic data, but is based on concepts used by *Xenon*

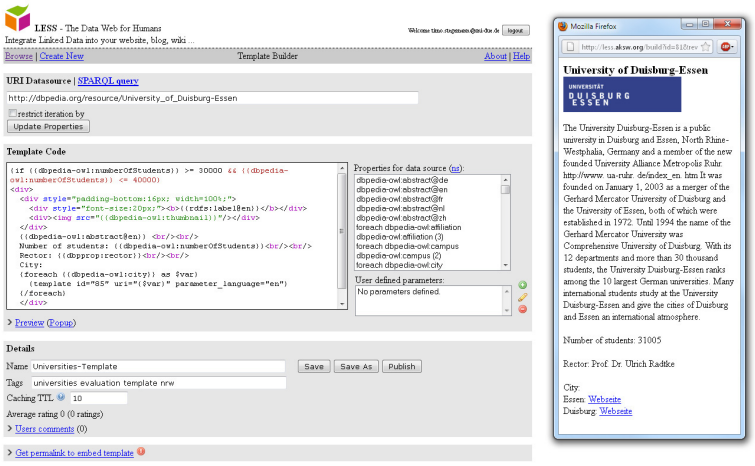
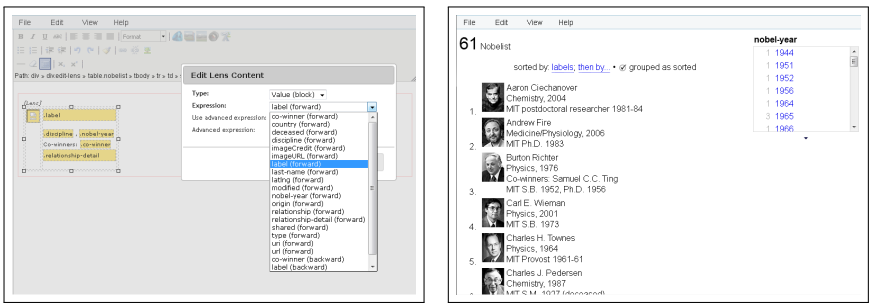


Fig. 1. LESS template editor (left) and the generated HTML output (right)

and *Fresnel*. *Dido* documents enable direct manipulation and creation of data as well as presentation directives. The embedded data is saved as key-value pairs in JavaScript-Object Notation (JSON). Selection and visualization are defined with lenses and views similar to *Xenon*. The user can directly choose in the editor, how the selected data should be displayed (Figure 2). For example, if the data should be represented as a list or table.



(a) Template (b) Document

Fig. 2. Dido's lens editor (left) and the generated result (rigth)

All approaches have in common, that they define their own templating languages for dealing with semantic data. Only *Dido* and *LESS* include editors facilitating the creation of new stylesheets. The editors focus on experts for semantic technologies, requiring a certain level of knowledge of the field. Editors for the other formats have not been developed so far.

X3S was designed to solve these shortcomings and provides a standard-based open format for semantic stylesheets. It can be used with arbitrary RDF-based data sources. The reference implementation provides an easy to use editor which enables non-experts the usage and creation of X3S documents.

3 The X3S Format

X3S is a format for describing semantic stylesheets. A semantic stylesheet is a template that can be applied to a RDF-based data source and leads to a filtered and styled subset of this data as a result. Its Goal is to transform semantic data into human-readable and exchangeable HTML documents. X3S defines a workflow consisting of the four steps *Querying*, *Restructuring*, *Transformation*, and *Styling*, most of which rely on established formats and technologies. Figure 3 illustrates the X3S workflow.

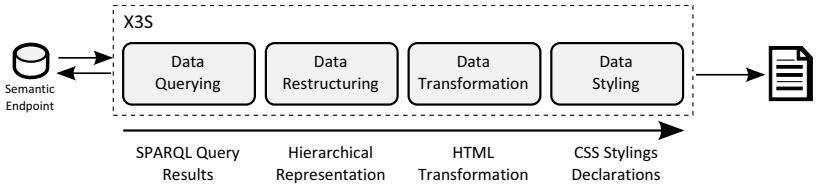


Fig. 3. X3S workflow to transform semantic data into a HTML document.

3.1 Querying

X3S uses SPARQL to retrieve content to be worked with. Any SPARQL compliant data source can be used such as *DBPedia*. With X3S, a single SPARQL query, that can request arbitrary properties, is deployed. The result of a SPARQL query is a set of property-value pairs, that are formatted in a W3C conform XML representation[3].

3.2 Restructuring

SPARQL results are not per se ordered hierarchically, leading to potentially redundancies. Thus, these results may have to be re-structured in order to remove redundancies and allow an unsophisticated way of processing the previously returned SPARQL-results. To achieve this, an additional XML-structure is used, onto which all SPARQL-results are mapped. Finally all result sets are aggregated into a single XML-structure, that contains all values as attributes of XML-nodes. Figure 4 shows a restructuring process.

[3] `http://www.w3.org/TR/rdf-sparql-XMLres`

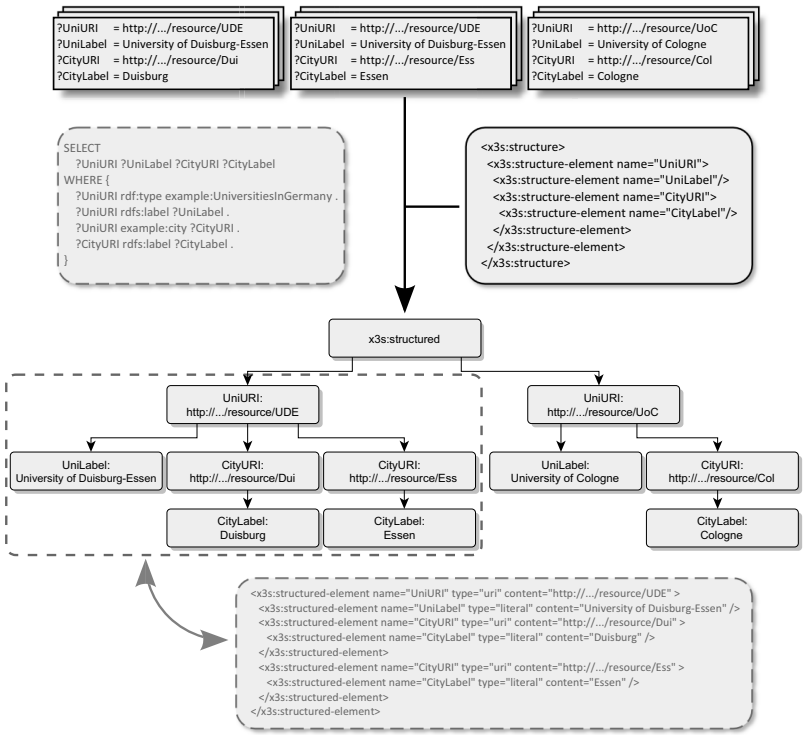


Fig. 4. SPARQL-results with redundancies are mapped onto a predefined hierarchical XML-structure, that meet the hierarchy of the primary SPARQL-request. All values are saved in the node's attributes.

In this example, information about German universities and their cities are requested from an notional SPARQL-endpoint. Because the University of Duisburg-Essen is located in two cities simultaneously – Duisburg and Essen – the SPALRQL-request returns two result sets with partial redundant data. The information about the university is equal, the information about the cities differs.

Direct processing of this data via XSLT, which is used in the next process step, would require some grouping mechanisms, that are possible but become very complex when nesting more than only one object into another. Therefore all result sets are mapped onto a predefined hierarchical XML-structure, that meets the hierarchy of the SPARQL-request without any redundancies.

3.3 Transformation

We use XSLT 1.0 for transforming the resulting XML trees or fragments into HTML, including CSS classes and IDs that can later be used to decorate the HTML (see Figure 5).

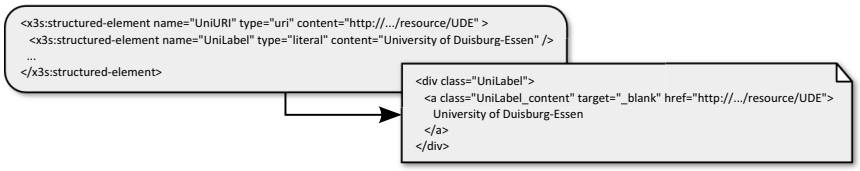


Fig. 5. The restructured Data is transformed via XSLT into an HTML document.

3.4 Styling

In order to specify the appearance of the documents, CSS can be embedded.

4 An Editor for Semantic Stylesheets

In order to create, edit, and preview X3S files, we developed an editor as a rich internet application based on Adobe Flex. The editor runs on any web browser with flash capabilities and can make use of existing semantic data repositories that provide a SPARQL interface.

Figure 6 shows the main window of the editor. In order to enable non-technical users to create and maintain stylesheets, we use point-and-click interactions instead of complex text-based directives: Drag-and-drop is used for selecting properties from a list of all possible attributes for the particular RDF class (Figure 6 A). The properties can be filtered and sorted by type, relevance[4], or in alphabetic order. In order to provide visual cues, datatype and object properties are displayed with different icons.

The user can drag elements from the list into the template located in the central workspace (Figure 6 B). When adding datatype properties to the template, an appropriate display format can be selected (for instance *depiction* in order to show the value of an URI as an image instead of the URI itself). As an additional feature, dynamic filters can be created based on datatype properties. This may be useful, if only a subset of all possible entities should be displayed (digital cameras with a price range from $100 to $200 for instance). Depending on the type of datatype property (String, boolean, integer, etc.), widgets such as sliders or date pickers for those filters are created dynamically.

The templates can finally be styled by using standardized CSS. The editor supports the user by directly offering graphical shortcuts to common CSS declaratives such as *bold* or *font size* (see Figure 7). Elements can have defined borders or margins, images can be adjusted in width and height, etc.

The elements in the workspace and the applied filters are internally used for generating a SPARQL query, which selects the respective result from the SPARQL source. This result is transformed into a hierarchical representation and transformed into HTML by the browser's XSLT processor (incorporating the CSS information). The results can be previewed in an IFrame (Figure 6 C), so that the user receives immediate feedback.

[4] *Relevance* is measured by the frequency of occurrence of this property among all instances of the class.

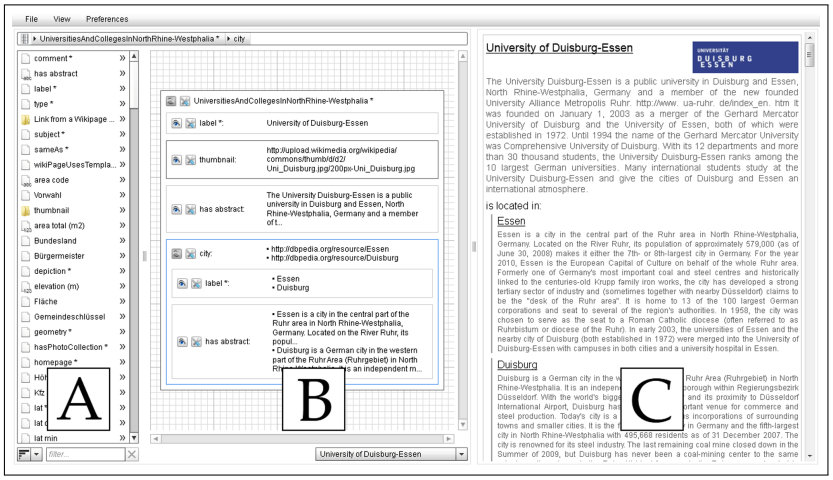


Fig. 6. X3S editor while creating a semantic stylesheet, working with data from DBpedia.

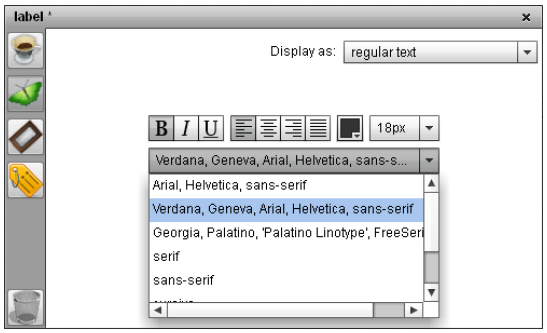


Fig. 7. Styling and formating options for regular text.

After the template has been created, it can be exported as an X3S file or archived as an HTML document.

5 Comparison

After introducing X3S and the corresponding editor, we now compare our approach to the ones described in the related work section (see Table 1). In terms of “Separation of Concerns”, all procedures but Dido are accessing their data from a separately saved source. Only X3S and LESS have a “SPARQL-Endpoint Support” and are able to request their data directly from an external semantic database. All approaches provide means for “Property Selection”, so only a subset of the data can be selected for presentation. “Filtering” of entries with regard to particular data values is also possible.

	X3S	Xenon	Fresnel	OWL-PL	LESS / LeTL Dido	
Separation of Concerns	●	●	●	●	●	○
SPARQL-Endpoint Support	●				●	
Property Selection	●	●	●	●	●	●
Filtering	●	●	●	●	●	●
Styling	●	●		●	●	○
Nested Object-Properties	●	○	○	●	○	
Usability (Developer)	●	○	○	○		
Usability (Author)	●	○	○	○	○	○
Usability (End User)	●	●	○	●	●	●
Templating	●	●	●	●	●	●

Table 1. Comparison of the approaches: ● support, ○ partial support, no entry: no support.

"Styling" of the data cannot be specified freely with Fresnel. All other candidates convert the data into HTML and can style them with CSS. It is simple with X3S and OWL-PL to display "Nested Object-Properties", that are completely integrated into a single stylesheet. With the other procedures, a stylesheet author has to build a separate stylesheet that must be embedded into the primary one. All candidates are able to re-use the stylesheets by a "Templating" mechanism.

One main goal for X3S was a high usability for developers, stylesheet authors, and end users. To reduce the amount of work for "Developers" to implement X3S, we tried to restrict X3S mostly to familiar and approved techniques. Xenon, Fresnel and OWL-PL however use own languages to describe the stylesheets, which are created with them. The "Author's" usability is principally determined by the tools and editors, with which the author can create a new stylesheet. LESS' and Dido's editor are capable to facilitate the author's work, but in our opinion are to complex to be used by non-experts in the field of semantic data. The "End User" should normally not even notice the use of semantic data. So most approaches generate a HTML-document from the template or semantic stylesheet or, such as Fresnel, require a dedicated browser to view the data.

6 Evaluation

We compared the X3S-Editor with the LESS-Editor in an evaluation and evaluated their respective usability. The essential results of this evaluation are presented in this section. The evaluation was designed as a user study during which each test user had to solve five tasks with both editors (Within subject design). All five tasks combined created a semantic stylesheet, where each task should cover an essential part of the whole stylesheet creation process – property selection (T1), data styling (T2), adding static data (T3), filtering (T4), and nesting further objects (T5).

In order to derive the efficiency of the editor, we measured the time needed to to complete a single task. To find out it's effectiveness, we recorded whether a test user

solved a task, needed assistance to solve a task, or could not solve a task. To get information about the user's satisfaction, the test users had to rate ten statements on a 5 point Likert scale after solving all five tasks with an editor, defined by the System Usability Scale (SUS) [2].

Ten test users participated in this study. Two users stated that they have well HTML and CSS skills. The others described their skills as rather weak. Six participants rated their general programming skills as above average, while a single participant had no programming experiences. The editors were preconfigured to work with DBpedia's SPARQL endpoint. The test users' global task was to create a stylesheet, containing information about universities and the cities in which they are located. No participant had used one of the two editors previously. Before the test started, the users were briefly instructed about the basic functionalities of the respective editors. They also received a summary of the LESS-Editor's commands, needed to solve the tasks.

Most participants were able to solve the five tasks with the X3S-Editor without any assistance. However, few test users needed assistance while solving tasks 4 and 5, because they did not find the proper filter settings or did not realize, that they can embed further objects into an already existing one. The LESS-Editor however confronted the test users with more problems. In most cases, the users needed assistance. Few test users could not even solve some tasks with repeated assistance. Only task 3 was simple enough to be solved by every participant. Because the LESS-Editor produced an error while inserting a property with a number value, most participants were not able to solve task 1. Only one user was able to repair this error without assistance. Most participants lacked the necessary CSS commands, needed to solve task 2. Since the LESS-Editor did not support the users when filtering different values, only the six participants, that rated their general programming skills as above average, were able to solve task 4. Even though the users had an overview of LESS' necessary programming commands, most were not able to embed a further object into the main stylesheet, tested in task 5. Figure 8 shows the results in detail.

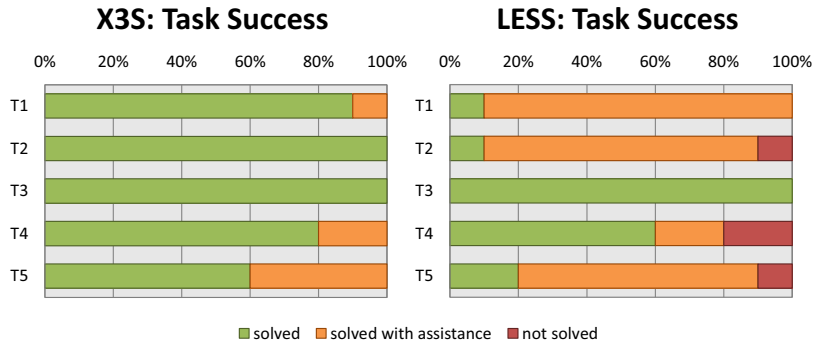


Fig. 8. We measured the both editors' efficiency by the user's task success. Most participants were able to solve all task with the X3S-Editor themselves. However, most test users needed assistance to solve the same tasks with the LESS-Editor.

In average the participants could solve most tasks significantly faster with the X3S-Editor than with the LESS-Editor (231% (T1) - 303% (T2) of the time needed with the X3S-Editor). Only Task 3 could be solved faster with the LESS-Editor. Summed up, the test users needed 7:26 minutes on average to create the whole stylesheet with the X3S-Editor. With 16:36 minutes on average, it took twice as long to do the same work with the LESS-Editor. All task times are shown in Figure 9.

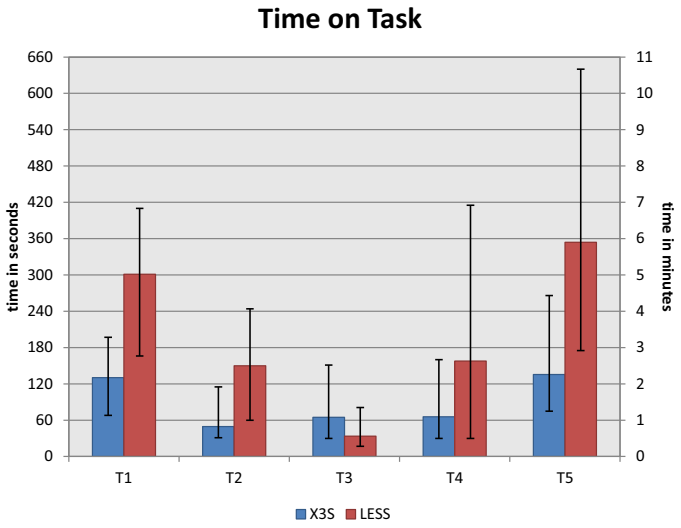


Fig. 9. Average times to solve the respective task. The error bars represent the minimum and maximum time, needed to solve a task.

Figure 10 shows the results of the System Usability Scale. The X3S-Editor achieved an average value of 90.5. One participant even rated the X3S-Editor with the maximum value of 100. However, the LESS-Editor only achieved an average value of 27.5. An average SUS score under about 60 points can be called as “relatively poor”, while a score over about 80 points could be considered “pretty good” [9].

7 Discussion

In this paper, we introduced X3S as well as a corresponding editor. X3S is a specification for filtering and displaying semantic data. The corresponding editor allows the creation of templates for data exploration and visualization, based on this X3S specification. X3S supports the separation of data and layout, arbitrary filtering of data and any kind of visualization via CSS. One main focus in the development of X3S was the ease of use for developers and end users.

The presented technique provides a variety of ways to use semantic information in various web applications. For example, predefined X3S templates in different config-

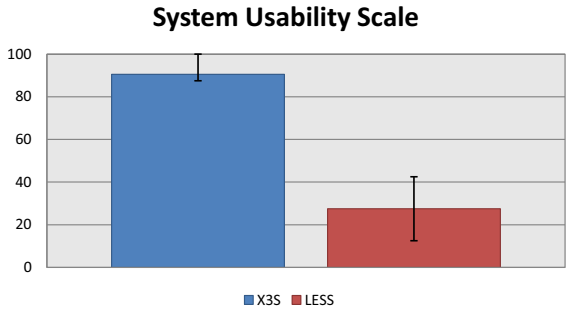


Fig. 10. Results from the System Usability Scale. The X3S-Editor achieved an average value of 90.5. One participant even rated the X3S-Editor with the maximum value of 100. The LESS-Editor only achieved an average value of 27.5.

urations can be used, helping to search a product. Due to their ease of use, they are suitable to build environments for intuitive exploration of semantic data such as customer advisory systems based on multi-touch surfaces.

The evaluation showed that the X3S-Editor is able to achieve the desired goal, to allow all users – even non-experts in the field of semantic data and web design – to create semantic stylesheets. None the less we revealed functions, that can be optimized. Especially the combination of dynamically retrieved semantic data and statically added text can be improved. Our future work tends to provide a generic component for processing X3S and to test it in the context of different application systems.

References

1. Sören Auer, Raphael Doehring, and Sebastian Dietzold. Less – template-based syndication and presentation of linked data. In *ESWC '10: Proceedings of the 7th Extended Semantic Web Conference*, number 6089 in Lecture Notes in Computer Science, pages 211 – 224, 2010.
2. John Brooke. SUS: A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability evaluation in industry*. Taylor and Francis, London, 1996.
3. Matt Brophy. Owl-pl: A presentation language for displaying semantic data on the web. Master's thesis, Lehigh University, 2010.
4. Martin Hepp. Products and services ontologies: A methodology for deriving owl ontologies from industrial categorization standards. *International Journal on Semantic Web & Information Systems (IJSWIS)*, 2(1):72–99, 2006.
5. Martin Hepp. Goodrelations: An ontology for describing products and services offers on the web. In *EKAW '08: Proceedings of the 16th international conference on Knowledge Engineering: Practice and Patterns*, volume 5268 of *Lecture Notes in Computer Science*, pages 329–346, 2008.
6. David R. Karger, Scott Ostler, and Ryan Lee. The web page as a wysiwyg end-user customizable database-backed information management application. In *UIST '09: Proceedings of the 22nd Annual ACM Symposium on User interface Software and Technology*, pages 257–260, New York, NY, USA, 2009. ACM.

7. Emmanuel Pietriga, Christian Bizer, David R. Karger, and Ryan Lee. Fresnel: A browser-independent presentation vocabulary for rdf. In *ISWC '06: Proceedings of the 5th International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 158–171. Springer, 2006.
8. Dennis Quan and David R. Karger. Xenon: An rdf stylesheet ontology. In *WWW '05: Proceedings of the 14th International World Wide Web Conference*, 2005.
9. Thomas Tullis and William Albert. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Morgan Kaufmann, San Francisco, CA, USA, 2008.