Semantic Web Based Context-Adaptable Generation of Product Specific Documentation

Andrei Miclaus
TECO, Karlsruhe Institute of Technology (KIT)
miclaus@teco.edu

Till Riedel
TECO, KIT
riedel@teco.edu

Jack Unseld
TECO, KIT
unseld@teco.edu

Michael Beigl
TECO, KIT
michael.beigl@kit.edu

ABSTRACT

As users and developers have started to put the Internet of Things to good use, the approach of documenting applications has not evolved to handle the created complexity. As items, devices and systems become more customizable and adapted to their users, their documentation still lags behind. In particular, documentation covering the contextual behaviour and specific configuration of artifacts is needed.

We design a system that leverages semantic web technologies to create smart documentation on the basis of model based system descriptions and heterogeneous data sources, which are needed to create valuable and up-to-date documentation.

Based on two scenarios we show the benefits for both the development cycle and the user experience of Web of Things applications. The paper presents a mashup of Internet of Things, model driven development, semantic web and HTML5 MVC technologies for generating context-sensitive documentation.

1. INTRODUCTION

An increasing number of smart, mashable and customizable devices reach the market. While traditionally, systems were built from single vendor components, more and more end-users create their own systems from heterogeneous devices and software to fulfill their needs. The final product is thus created by the user.

Unfortunately, in such complex interaction patterns even a single component becomes increasingly difficult for a human to understand and configure. Taking the example of end-user installed home automation systems it can be shown that the overall conceptual model plays an important role in user acceptance when setting up such systems [3]. Subsequently, documentation packaged with the devices would need to provide appropriate information. However, today's documentation cannot take the current application context and user experience into account and thus fails to provide appropriate support [2].

Furthermore, product manufacturers also need to deal with the increasing complexity of the product documentation when delivering pre-customized items. Often they burden users to make sense of a documentation describing entire product families instead of the product variant they acquired.

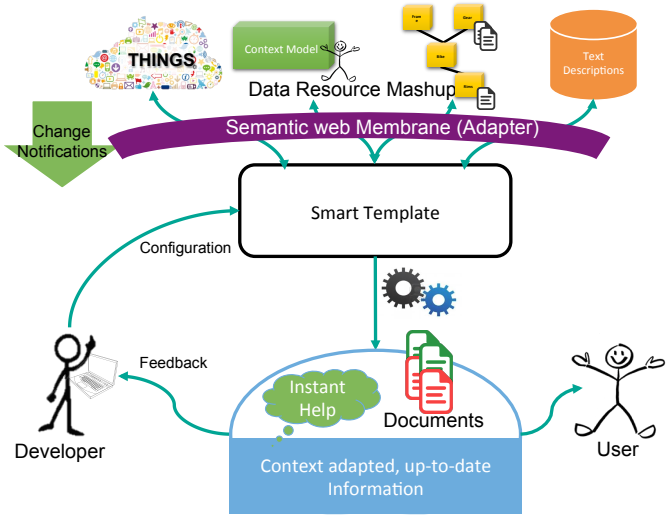

Figure 1: Interaction with the context-adaptable document generation system.

To address this problem, we envision a document generation system that creates a data mashup from different resources according to smart documentation templates. As seen in Figure 1, the developer configures the smart template instance, providing data source locations for a specific product. The smart template in turn queries the data sources using semantic web technologies and generates the up-to-date documentation. Using notifications, re-generation of the documentation can be automatically triggered when underlying data sources change. A recent study showed that such document generation may bring a further benefit: users will read the documentation more carefully than a static one

if they are told that it is personalized and suited to their current needs [12].

The presented system is designed to create personalized, specifically tailored system documentation. This allows domain experts to control the information passed onto the user, whilst allowing an automated update of the data contained in the document based on system models.

In this paper we present the scenarios from which we derive the requirements and describe the underlying system architecture and its interfaces to the human and machine users. For this we adhere to WoT practices [6] by employing RESTful services that exchange semantic enhanced data like JSON-LD. We show how different existing technologies can be combined to create smart documentation mashups.

2. RELATED WORK

Supporting users in their understanding of the system is important. How a user perceives a system is dependent on the presented information that helps form the conceptual model in the user's mind. This conceptual model plays a key role in installing and using such a system and subsequently influences the user experience [3].

A recent study in a home automation scenario evaluates how participants respond to generated documentation when confronted with a home automation system installation [12]. The authors, however, focused on the study without providing technical details as to how the documentation is generated. In this paper we wish to describe the design and implementation of the concept and describe how this system can generate the installation documents. The study also showed that users belonging to a younger generation mostly ignore description documents and wish to rely on their smartphone to interact with systems [12]. Providing documentation in multiple modalities and at the right time on their devices may speed up the understanding of the system and improve the user experience.

Providing automated assistance in the augmented reality domain, the augmented reality manual automatically adapts to the user and the current state of the installation [14]. The authoring process is automated by using video recordings of a physical task. However, textual descriptions which can aid in the understanding of the task need to be manually added. Using our system, a repository may be queried to supply the automatic authoring process with additional textual documentation or images based on the currently identified task.

Industrial software product lines present a similarly difficult challenge for documentation because of their high degree of complexity and customization. The DOPLER approach aims to support the creation of documents by using flexible product line variability models [5]. The tool focuses on aggregating knowledge into the system for generation purposes. However, we wish to allow the composition of arbitrary sources in the document generation process and allow for easier changes in the domain because the system we envision does not rely on a unified model.

The Darwin Information Typing Architecture (DITA) [8], is an XML-based, architecture for authoring, producing, and delivering technical information for industrial projects. A commercial implementation is Arbortext[1]. Although it provides a solution for document generation, data has to conform to a certain structure, whereas our approach focuses on unified data retrieval and composition without having to change the system models. In this way we address the needs of constantly changing heterogeneous domains like the web of things. Furthermore the approach presented in this paper allows reasoning on the data, thus bringing smartness to the generated documents.

Model-driven documentation tools like EMFDoc[2] can be used to add documentation to software models. Each model element in a referenced XMI file can be annotated with a comment, which is automatically added as GenModel annotation. In addition, EMFDoc computes the coverage of the documentation and shows the results in the outline view.

In the quest for the web of things, researchers have already integrated semantic technologies within IoT applications [10]. By using ClickScript[3] the users can customize their applications to their liking. The document generation system envisioned in this paper benefits from these advances as more and more resources can be queried semantically out of the box without the need for adapters.

In accordance to the vision of creating a mashable World Wide Web of Things [7], a documentation generation system that combines several data sources using semantic technologies seems a natural step towards a unified experience.

Web technologies have gained versatility in the last years. Maintaining server side code and sending it to the client as needed, adds great flexibility to standalone HTML5 web pages. Incorporating existing template engines and semantic web libraries is thus achievable using for example Node.js[4] in combination with libraries such as Browserify[5].

3. USE CASES

In order to ensure our solution maintains real world relevance, we derived requirements from the challenges present in the domains of configurable things and configurable Internet of Things presented below.

3.1 Configurable Things

To illustrate the problem of creating documentation for reconfigurable things based on state of the art model driven configuration technology, we look at a simple Bike configurator demo by Configit[6]. This configurator allows for the definition of millions of bike variants and acts as representative for high variability industrial products we want to deal with [13].

A product, the bike, requires several document types, each focused on different aspects tailored to the specific variant they describe (see Figure 2). The technical sheet for a bike

[1]http://www.ptc.com/product/arbortext
[2]DevBoost EMFDoc, http://reuseware.org/index.php/
[3]ClickScript, http://clickscript.ch
[4]Node.js, http://www.nodejs.org/
[5]Browserify, http://browserify.org/
[6]Bike Shop, http://demo.configit.com/BikeShop2/Default.aspx

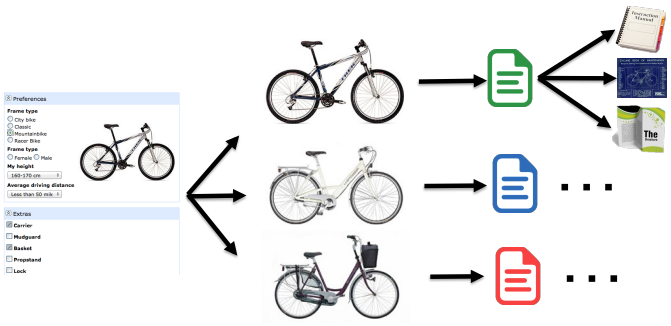

Figure 2: From the online configurator to the different variants of a bike and the documentation types accompanying each of them.

usually contains the description of the individual parts of the bike, one after the other. A component may have different parameters or attributes that also need to have attached explanations. A brochure on the other hand, may contain text more suited for marketing purposes. The manual may contain assembly or maintenance instructions alongside the information from the aforementioned documents. Although each of these documentation types describe the same product, large parts of the contents may vary, while some are reused. Handling the composition in an automated manner allows the inclusion of content management systems that can handle text administration in an organized manner.

While all parts and the structure should be included in the documentation depending on the situation, mentioning non-existent parts of a bike may confuse and decrease the user experience of the product. Furthermore, a bike may be sold in different countries with different preferences. In this case, not only the language of the documentation varies, but also the technical configuration, and the cultural background and context of use.

Often small configuration changes, may entail differences in structure that are known to the expert assembling the bike. Depending on preferences, different frame parts might hold the cables, breaks etc. . This way documentation shows its importance when the usefulness of a thing is assessed by a user. Users might find aspects that they do or do not like in a certain model. Furthermore, documentation can aid the domain expert in the discovery of inconsistencies. Quality documentation is a good indicator for the usability of the thing itself.

Given the requirements derived from this scenario and the large variant space one can easily see that manual creation of product specific documents is not feasible in such a situation. Even if the number of variants is small, manually creating documents for one such variant still remains an error-prone and time intensive task. Therefore, in practical scenarios, customers are usually provided with a generic documentation[7].From the perspective of the user, generic documentation can be time consuming to read and may cause confusion because of inconsistencies with the product at hand, thus hindering the comprehension of the system.

3.2 Configurable Internet of Things

While the bike shop example shows some basic problems that we have to solve for the documentation of things, in an Internet of Things we want to cover complex systems comprised of a variety of products. Rather than looking at a monolithic reconfigurable thing, we are looking at mashups of (preconfigured) things representing a downloadable application. One of our target use cases is situated in the domain of home automation [12].

In the mentioned use case, the openHAB[8] framework is used by the authors to facilitate the device interoperation. It is an integrating middleware that provides device interconnectivity and the possibility to specify behavioural rules across a multitude of devices. A developer provides, via app store, the logic in form of openHAB rule models and a template describing the application. When writing the template we supply an editor to the domain expert that can tap into the code model and provide content assistance.

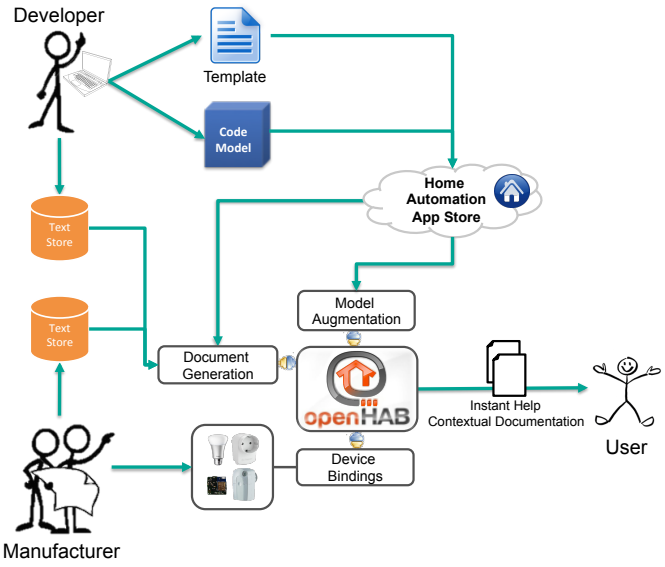

Figure 3: Home automation scenario where documentation is provided by developers and manufacturers alike, while the users receive only the relevant parts.

An app download from the store provides templates and augments the openHAB models (see Figure 3). The device manufacturers can add textual descriptions of arbitrary detail to a text store. The text store can provide the data as answers to semantic queries. Furthermore, the current state can be queried from the openHAB runtime models.

The different manufacturers may annotate things with tags that the app creators can use in documentation templates. E.g. using a thing as light sensor will trigger the inclusion or linking of the text describing the light sensor functionality. Especially when being confronted with heterogeneous

[7]The interconnected car Opel Adam is one of few customizable products which ships with personalized documentation, http://www.opel.com/microsite/adam/

[8]OpenHAB, http://www.openhab.org

components, as seen in Figure 4, the user is burdened with making sense of it all. Adapted documentation can improve the identification and composition of the things forming the system and ease the task of the user.

Figure 4: Packaged devices as they are delivered to and unpacked by customers after acquiring them from different manufacturers.

As openHAB relies on EMF technology, the textual descriptions of the model elements can be managed on the same technological level using EMFDoc. This base documentation may then be uploaded to a text store to facilitate translations or variations for different documentation types.

Not only can the app developers provide documentation for the users, but the openHAB developers themselves can provide documentation for the developers using the framework, either in form of standalone documents or by providing instant help, say as JavaDoc snippet or contextual help, while developing their apps within the openHAB designer.

Like the Configit configurator the system is model based. Device and rule management are based on the Eclipse Modeling Framework (EMF). OpenHAB can interface with devices via bindings to bridge the gap between proprietary protocols and executes the model within its OSGi runtime, thus integrating the smart devices on the technical level. We wish to augment this integration by providing the user with unified documentation adapted to his or her use case.

4. DESIGN AND IMPLEMENTATION

The first goal of our document generation architecture was to create a reusable system that integrates with both use cases mentioned and their underlying model driven development schemes. Furthermore, we want to enable the use of existing heterogeneous information to create documentation. As the documentation is a mashup of textual descriptions linked to the underlying system model it is possible to provide instant help documentation snippets to be used in digital interface such as augmented reality or smartphone and tablet based applications, as well as integration with IDEs to provide development assistance.

The second goal is to integrate with a heterogeneous web-based infrastructure. E.g. to handle internationalization challenges and different use cases for product documentation, we want to be able to connect to text storage similar to a content management systems and wikis. This allows handling fragmented textual descriptions for components. Text block inclusion in the documentation is filter based using meta-data tags added during the authoring of the texts.

Beyond the engineered product model, the information contained in the product knowledge base is likely to impact the contents of the documentation. The need for manual document updates immediately bring the product and its documentation out of sync. Furthermore, individual users may find certain types of information more appropriate or require a different detail level to suit their needs and qualifications.

4.1 Documentation Generation

Several components need to provide information to the generator, for the document generation to take place: the document structure, the style information, the text blocks and the concrete system variant for which the document is built. According to the model these input resources are federated and semantically linked to obtain the final document.

In the likely event of changes to underlying system describing information during the development or maintenance process, all documents listening to these sources will be notified and a re-rendering of the template takes place.

A typical software product has a knowledge base consisting of several resources which we will further call source models. These source models are the basis onto which the product is created. These source models can be anything ranging from UML Models or custom models and meta models to Excel spreadsheets and property files containing product specific information (see Figure 5).

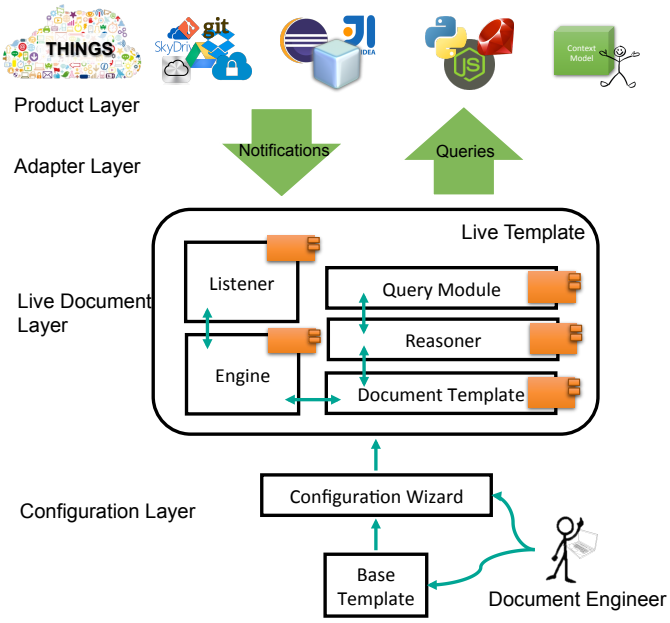

Figure 5: Document generation system architecture overview including heterogeneous data sources.

To combine the data from the different source models, we use semantic web technologies such as JSON-LD, OWL and SPARQL, even allowing the integration of reasoning technologies. SPARQL endpoints for each type of source model

in the description of the product answer the queries sent from the smart template. By separating the translation of model data into a separate layer, we allow independent changes or addition to the source models.

To handle the semantic context inside our web applications after it has been queried, we use JSON-LD[9]. This entails high flexibility in terms of programming support and combined with HTML5, the possibility arises to incorporate complex logic into standalone web pages (such as intelligent help sites for product features). SPARQL's [15] ability to query semantic stores via REST endpoints and the built in mechanism to filter searches make it a good option to query bigger semantic stores.

The federated source models provide up-to-date information if they are alive during the execution. This is the case with the openHAB framework, where updates to the models change the execution environment. This is an especially useful feature of the framework that allows truly live documentation to be generated, which reflects the state of the system at the time of reading. The SPARQL endpoints for the source models are declared in the product specific part of the documentation, as they are a direct declaration of the data sources the product is made of.

In addition to the use of existing models and federated information we create a context model. The context model is currently created manually within the documentation component and contains meta-data used to adapt the generated documentation not only to the specific variant of the product but also to the current user or other external context information.

The federated knowledge contained in the source models is valuable for helping the user create a better mental model. Through integration it is possible for a documentation developer to create a single personalized narrative of the product.

4.2 The Model Bridge

While relying on existing components for most parts, we are implementing a light-weight java-script document generation framework. Rendr[10] allows both static and dynamic text rendering on a server as well as in a web browser based on a strict model-view-controller principle.

As described above, the model is made up solely by web endpoints that are combined based on a HATEOAS[11] principle using JSON-LD. In our authoring workflow we are currently only supporting the linking of models in the web-browsers but also plan on supporting link-discovery frameworks (like LDIF[12]) as well as server-side reasoning (via OWL and/or SPIN rules) employing user defined semantics.

The core of our specific architecture implementation comprises a model bridge for the application models inside the eclipse framework. For this we are including a light-weight web-server as Eclipse plugin that exposes the EMF models as linked data. Our current implementation is based on the EMFTriple[13] and the Fuseki Webserver from the Apache Jena Project[14] incorporating a Pellet reasoner[15]. The documentation fragments we are currently handling inside of drupal via its ARC2[16] based RDF Endpoint. Using this simple set of technologies we were able to implement a proof of concept for both use cases. However, the set of components provides a basis for more advanced use. This setup allows the implementation of context-sensitive HTML5 help inside the OpenHAB framework that acknowledges the usage context of a thing inside a complex workflow based on a single ID.

The architecture is further flexible enough to generate other kinds of documentation. In order to generate documentation that allows for further manual editing, we are using Pandoc[17] to generate formats such as word documents.

Figure 6 shows some static standard documentation and its generated counterpart in comparison. The documentation on the right exclusively contains information blocks relevant for a thing as light sensor (highlighted in red). Leaving out unused features of multi-purpose devices greatly reduces the size of the documentation making it more concise.

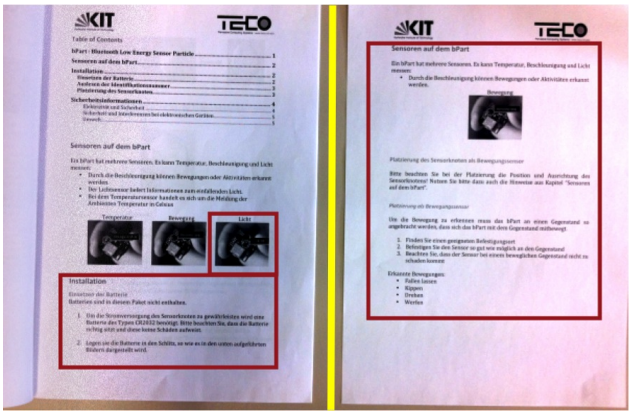

Figure 6: Comparison of standard (left) and generated documentation (right).

5. DISCUSSION

Other research shows that it is possible to generate natural language text directly from system models like UML Diagrams, to provide non-technical users access to the contained information [11, 4]. While these approaches do not focus on generating documents, they can provide valuable data sources for the document generation process described in this paper and allow fully automatically document generation without human text authoring.

Explicitly managing the textual descriptions of the system components allows external agencies to access the textual data easily, in an organized manner to proof read or translate it, providing implicit versioning support and consistency

[9]JSON for Linked Data, http://json-ld.org/
[10]Rendr, https://github.com/rendrjs
[11]Hypermedia as the Engine of Application State
[12]LDAP Data Interchange Format
[13]EMFTriple, https://github.com/ghillairet/emftriple
[14]Apache JENA, http://jena.apache.org
[15]Pellet, http://clarkparsia.com/pellet/
[16]Drupal ARC2 SPARQL, https://www.drupal.org
[17]Pandoc, http://johnmacfarlane.net/pandoc/

checks. This can also foster the collaboration among developers, marketers and other contributors, allowing a separation of concerns during the documentation development.

Documentation creation in enterprise environments requires the adoption of documentation standards. Implementing standards like the IEEE Software Documentation Standard [1] requires tedious manual work, if no automated infrastructure exists. Additionally, techniques such as structured writing can be more easily integrated into the documentation development cycle [9].

In our research and implementations we have only considered the generation systems as well as the end-user interface to configuration and documentation while making assumptions about the authoring process. For a system to scale, especially those parts still need to be addressed.

6. CONCLUSION

In this paper we presented a system concept for the generation of product specific documentation that is applicable to physical and non-physical system families. By allowing a data mashup with a unified semantic web querying interface between the diverse source models a product may have, we enable contextual documentation generation.

As the documentation is generated from the underlying system models, it follows naturally that a future version will allow for two way synchronisation enabling running software modifications by altering presented documents. As a results, user-to-system dialogue becomes possible.

When implementing feature requests, developers usually create or modify parts that are visible to the customers. Providing instantaneously generated documentation adds yet another feedback loop in the development process. This feedback loop is not only meant for developers or technical personell, but can be used by arbitrary stakeholders. Managers, marketers and sales people can receive instant updates when the code is committed to the central repository and are able to provide valuable input to the development team, preventing costly changes later in the product cycle.

But as with all new concepts and tools, real world usage may vary. Nevertheless, having a higher quality documentation that corresponds to what the customer can use effectively is an added benefit worthwhile and relieves employees of repetitive and error-prone tasks.

We believe that many applications can benefit from such a system because it is accompanying rather than intrusive with regard to the development process and provides a feedback on the state of the system after changes take place.

7. ACKNOWLEDGMENTS

This work was partially funded by the German Federal Ministry of Education and Research (BMBF) as part of the InstaGuide project (grant number 01IS12051).

We wish to thank the colleagues at Siemens CT Austria for their insights and support.

8. REFERENCES

[1] IEEE SA - 26514-2010 - IEEE Standard for Adoption of ISO/IEC 26514:2008 Systems and Software Engineering–Requirements for Designers and Developers of User Documentation.

[2] S. Antifakos, F. Michahelles, and B. Schiele. Proactive Instructions for Furniture Assembly. In *Proceedings of the 4th international conference on Ubiquitous Computing*, UbiComp '02, London, UK, UK, 2002. Springer-Verlag.

[3] C. Beckmann, S. Consolvo, and A. LaMarca. Some Assembly Required: Supporting End-User Sensor Installation in Domestic Ubiquitous Computing Environments. In *UbiComp 2004: Ubiquitous Computing SE - 7*, volume 3205 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004.

[4] H. k. Burden and R. Heldal. Natural language generation from class diagrams. In *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation - MoDeVVa*, page 1, New York, New York, USA, Oct. 2011. ACM Press.

[5] D. Dhungana, R. Rabiser, P. Grünbacher, K. Lehner, and C. Federspiel. DOPLER: an adaptable tool suite for product line engineering. *11th International Software Product Line Conference (SPLC 2007)*, pages 10–14, 2007.

[6] D. Guinard, V. Trifa, F. Mattern, and E. Wilde. From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices. In D. Uckelmann, M. Harrison, and F. Michahelles, editors, *Architecting the Internet of Things SE - 5*, pages 97–129. Springer Berlin Heidelberg, 2011.

[7] D. Guinard, V. M. Trifa, and E. Wilde. *Architecting a mashable open world wide web of things*. ETH, Department of Computer Science, 2009.

[8] N. Harrison. The Darwin Information Typing Architecture (DITA): Applications for Globalization. pages 115–121, 2005.

[9] R. E. Horn. Structured Writing as a Paradigm. 1998.

[10] S. Mayer and N. Inhelder. User-friendly configuration of smart environments. *Pervasive Computing . . .*, pages 163–165, 2014.

[11] F. Meziane, N. Athanasakis, and S. Ananiadou. Generating Natural Language specifications from UML class diagrams. *Requir. Eng.*, 13(1), Jan. 2008.

[12] A. Miclaus, T. Riedel, and M. Beigl. End-User Installation of Heterogeneous Home Automation Systems Using Pen and Paper Interfaces and Dynamically Generated Documentation. pages 1–6. The 4th International Conference on the Internet of Things (IoT 2014), 2014.

[13] J. Mø ller, H. Andersen, and H. Hulgaard. Product configuration over the internet. *Proceedings of the 6th INFORMS*, 2001.

[14] P. Niels. Intelligente Augmented Reality Handbücher Zeigen, wie's geht - Werkerunterstützung für die Fabrik der Zukunft, 2013.

[15] E. Prud'Hommeaux, A. Seaborne, and Others. SPARQL query language for RDF. *W3C recommendation*, 15, 2008.