Linguistics-Based Modeling Methods and Ontologies in Requirements Engineering

Florian Lautenbacher, University of Augsburg, Germany

Bernhard Bauer, University of Augsburg, Germany

Tanja Sieber, University of Miskolc, Hungary

Alejandro Cabral, Oracle Strategic Program, Argentina

ABSTRACT

Developing new software based on requirements specifications created by business analysts often leads to misunderstanding and lack of comprehension, because of the different backgrounds of the people involved. If requirements specifications instead have a clearly defined structure and comprehensive semantics, this obstacle can be resolved. Therefore, we propose to structure the requirements specifications using existing linguistics-based modeling methods and annotate the used terms with ontologies to enhance the understanding and reuse of these documents during the software engineering process.

Keywords: Ontologies, Requirements Specification, Semantic Data Model, Semiotics, Structural Modeling

1 MOTIVATION

Not only normal software development, but also the upcoming research area semantic-based software development (de Cesare, 2007) typically has an iterative software development process starting with the requirements engineering and requirements analysis phase. Before beginning with the development of software, the needs of the customer must be clarified and summarized into requirements specifications. These requirements contain all (or nearly most) of the details about the software product to be developed and are normally described in natural language. Some companies have therefore defined style-guides. However, most of the used terms are not defined in a concrete way which leads to misinterpretation and incomprehension, i.e. the semantics are not defined clearly. Sometimes glossaries are used to describe the expressions, but even those can be interpreted differently by various readers/writers. Missing or not clearly defined requirements lead to change requests for the software product once it is tested or, in the worst case, when it is used by the customers. The customers might have thought of

DOI: 10.4018/jeis.2010120202

something different, but their requirement has not been described properly in the requirements specification. Therefore, it is critical to specify the requirements as precisely as possible in the first place to avoid unnecessary changes to the finished product afterwards and to build the product on time and in budget.

As stated in Rupp (2006), software (S) is a combination of documentation (D) and code (C), i.e S=D+C. The documentation should not only cover the source code and its comments itself, but also the description of using the product afterwards (software documentation), any kind of technical specification and documentations, like functional and non-functional aspects, UML diagrams or database descriptions, etc. In document engineering, which is concerned with these issues, internal and external document engineering can be distinguished. The former refers to the documentation produced during the whole software development process, while the latter refers to the documentation produced for the system's users after the product is released (Rueping, 2003). Requirements specifications can be seen as a typical example of internal documents, whereas user manuals are typical external document examples. There are some linguistics-based modeling methods that are widely used in external document engineering, which could also be used for internal document engineering, e.g. for gathering requirements. Using these modeling methods, the structure of documents and their underlying dependencies can already be reflected in the modeled segmentation of the documents, making it easier to be derived and annotated with semantic data afterwards. This semantic annotation is based on ontologies and can be used to describe the meaning of the constructs in a way that computers can not only read but also interpret.

We will therefore show how the semantics of requirements specifications can be gathered using linguistics-based modeling methods and that an annotation of these documents with ontologies can foster reuse and personalization.

This article is structured as follows: in the next section we describe the challenges of current documents and the difference of understanding some data between sender and recipient. Additionally, we describe our definition of data and how the communication between different persons takes place. Afterwards, we show how different linguistics-based modeling methods can be used to clarify the underlying meaning of terms. We evaluate several linguistics-based modeling methods and show a summary of our evaluation. We then use an example to clarify the usage of the modeling methods as well as introduce the process and benefits of semantic annotation through the usage of ontologies. Subsequently, we show some related work before we conclude describing the benefits of using linguistics-based modeling methods and ontologies.

2 CHALLENGES OF SEMANTIC REQUIREMENTS ENGINEERING

In this section we introduce the basics of linguistics such as the Speech act theory, before we introduce models for the description of data and the process of communication that are required for understanding the problems in Requirements Engineering and possible solutions.

Speech Act Theory

John Langshaw Austin developed his Speech Act theory in such a way that today, more than 45 years later, we find it useful to conduct our research on semantics in requirements engineering with reference to the theory. In Austin (1962) he introduced an informal description of the idea of an illocutionary act that can be captured by emphasing that when we use language as more than a mere way to state things as true/false, we actually do the action being pronounced or denoted. A good example is when a minister joins two people in marriage saying: "*I pronounce you husband and wife*".

To further explain this theory, Austin declared three types of speech acts:

- **Locutionary acts:** Saying something (the locution) with a certain meaning but not necessarily building a speech. It may be a word, sentence or sound. There are three different kinds of locutionary acts: It is at one level the production of certain noises and as such it is called the *phonetic* act; through the production of those noises the speaker produces words in syntactic arrangements and this act is called *phatic* act. Finally, through the production of words in syntactic arrangements, with certain intentions and in certain contexts, it conveys certain messages and is in this respect dubbed *rhetic* act.

- **Illocutionary acts:** The performance of an act in saying something, or basically the speaker's intent. John Searle developed further this category in 1969 (Searle, 1969) and identified five illocutionary points: *assertives* (true or false statements), *directives* (statements with a certain intent), *commisives* (statements which commit the speaker to a course of action), *expressives* (express the sincerity of the speech act) and *declaratives* (statements that connote a change of the world referred by representing as already changed).
- **Perlocutionary acts**: These acts have a direct effect on sensitive perception, feelings or actions of both the speaker and the receiver. These acts basically seek to change a state of mind, an idea or feeling towards a representation.

Austin's analysis and contribution through his Speech Act theory went beyond the referencial theories during his time and considered the context in which language was actually used. He was the first to consider the context and the listener as a part of the communication equation, relying on the concept of *convention* to depict an illocution-perlocution distinction. For Austin, illocutionary acts are based on the existence of convention, while perlocutionary acts are not. By this, Austin opens a window to another dilemma: *meaning*, though he fails to further develop his Speech Act in this area, thus allowing others to criticize his work:

Grice (1967), Austin's Oxford colleague, developed his own theory on meaning and distinguished *natural* from *non-natural* meaning, in terms of whether or not there exists a natural connection between an utterance and what is actually meant by it. Logically, non-natural meaning refers to those cases where this natural connection does not exist. In other words the meaning of any utterance consists in its intentional use by the speaker to accomplish his or her desire to get the listener to do something by revealing to him/her the actual intention the speaker has, and this cannot be solely based on the *convention* concept that Austin explained.

Similar to Grice's analysis Strawson (1969) criticizes Austin's theory as well, as he describes Speech Acts as not really dependent on conventions working as connections between utterances and meanings. He explains that a person can act without actually using an existing convention all the time in order to accomplish or perform an act by uttering something. Both Grice and Strawson acknowledge the presence of a deeper concept than the one Austin introduced when naming conventions: they both refer to *intention*. Strawson additionally rejects the illocution-perlocution distinction that Austin based on the existence of convention as a context identifier and integrator.

All this analysis though could not be complete without Wittgenstein's referential theory on meaning, where he presents the idea that language cannot consist of or be a linguistic rule of the signifier and the signified. If that were the case, there would be a space where another rule connects the statement of the rule with what it really signifies. Wittgenstein (1973) declares the true importance of *context* to determine the meaning. In order to really understand what an utterance means, the context needs to be present, considered and integrated with the language that is being used.

All these categories created by Austin and afterwards developed by Searle, Grice, Strawson and Wittgenstein apply to languages as we see them: understanding that by language we imply a system of communication consisting of sounds, words or characters used by two sources/ destinations to exchange information. In the case of a written exchange we can speak about a scribal act instead of a phonetic act, but the spirit behind the Speech Act theory remains the same, which makes it quite interesting for documentation engineering purposes and henceforth also for requirements engineering.

Illocutionary acts are performed with intentions. They are communicatively successful if the speaker's/writer's illocutionary intention is recognized by the hearer/reader. Illocutionary acts are all intentional and are generally performed with the primary intention of achieving some perlocutionary effect.

By sending certain data in the form of words or written commands we issue not just an order or perlocutionary act. The content of the statement is imbued with a certain meaning. In spoken languages this meaning can be implied by the tone/intonation used to communicate, or by different signs the speaker can send. Even if these are not used, a certain meaning can be transmitted if the speaker and the hearer know each other. The use of different codes and contexts help quite a bit: it is by them that a speaker constructs a statement, sends it as a message with an identified meaning, then a hearer receives it and using the codes already learned and identifying the context used to create this message, re-constructs the message received, imbuing it with another meaning that is analog to the one originally sent by the other source. In written documents this is much more difficult since the future reader is mostly not known and therefore the meaning must be made explicit in other ways. The challenges for written documents can be explained by using a semantic data model.

Semantic Data Model

When specifying requirements, these are summarized in some kind of requirements specification in a document containing *data*. But the term *data* is often used without any exact terminological definition. There are different definitions for this term and henceforth it is not used in a uniform way in papers and lectures. Therefore, we introduce our understanding of *data* following a semantic data model in order to describe the problems that occur between somebody who specifies the requirements (*sender*) and the person who reads the requirements specification (*receiver*).

According to the semantic data model introduced in Sieber and Kammerer (2006), data penetrates through various levels: it can have different *forms* (such as '13' and '***13***'), different kinds of *representation* (e.g. arabic numbers vs. roman numbers vs. textual description) and different *semantics* (for example the number '13' can describe the age of a person or the number of a house).

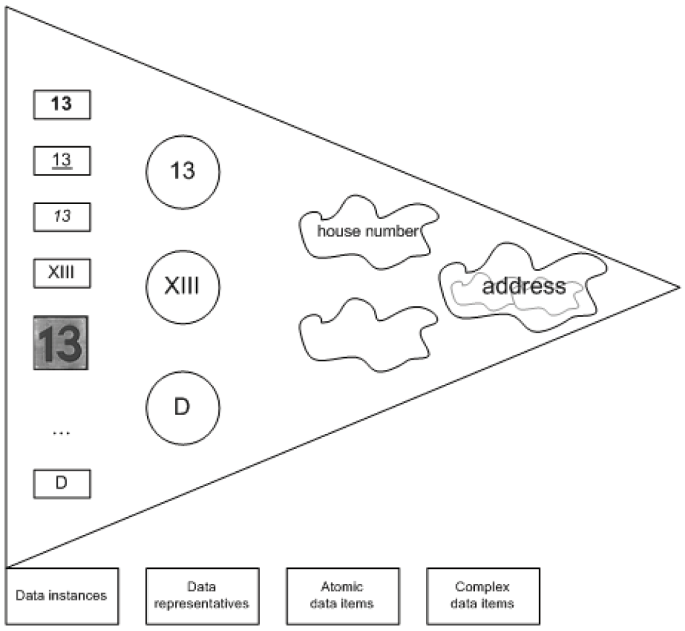
- At the level of form all data are called *data instances,* where a data instance itself is a semiotic entity in terms of Peirce. That is, it can appear as icon, index or symbol. The semantic data model bridges the existing gap between the understanding of data and the semiotics.
- At the level of representatives data appear in an abstract form and are then called *data representatives*.
- At the level of meaning data appear again in a more abstract version and are then called *data items*. These data items can even be split into smaller meaningful components (*complex data items*) or are not dividable (*atomic data items*).

Figure 1 shows the semantic data model in a graphical way: there are several data instances for '13' which belong to different data representatives and all to the atomic data item *house number* which might be part of the complex data item *address*.

Sender-Recipient Model

If someone wants to specify the *house number* as a data item, then it can happen that some-

Figure 1. Semantic data model (according to Sieber & Kovács, 2005) for the data item "house number"



body else sees the concrete data instances, but understands something different by it. Henceforth, it does not happen at the same level of knowledge. Data instances are the only part of the semantic data model which appear outside of a human being in a concrete form (as icon, index or symbol), all other levels of abstraction are intra-personal. Following this understanding of data the consequence is that meaning or semantics is also to be understood as something intra-personal and we are captured in our lingual possibilities to talk about it.

Every communication based on data instances between two parties has to consider that each person has a different background and different knowledge (compare Figure 2) and will derive from a data instance maybe different meanings. Therefore, the data instances have to be specified in a way the recipient can understand what they are meant for. If the sender wants to specify the number of a house and the recipient only recognizes the data instance '13', then he might think of it as the age of a person (compare Figure 3).

Semantic Communication Model

To describe what actually happens when a message is exchanged between different persons, we developed a semantic communication model considering existing communication models such as Shannon and Weaver (1964), Berlo (1960), Bühler (1965), Schramm (1954) or Flensburg (2007).

Therefore, we relate the Sender-Recipient model with the semantic data model and combine them using *messages*. In principal, each person has a different background and knowledge. Each time a document (e.g. a requirements specification) is written, the background and knowledge of the target recipients have to be considered in order to enable the recipient to understand the document. In spoken language this is quite easy since the recipients are (nor-

Figure 2. Sender-Recipient model of Sieber & Kovács (2005)

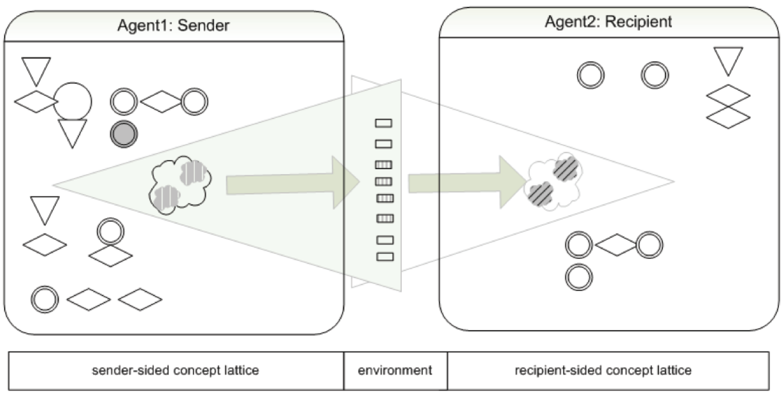
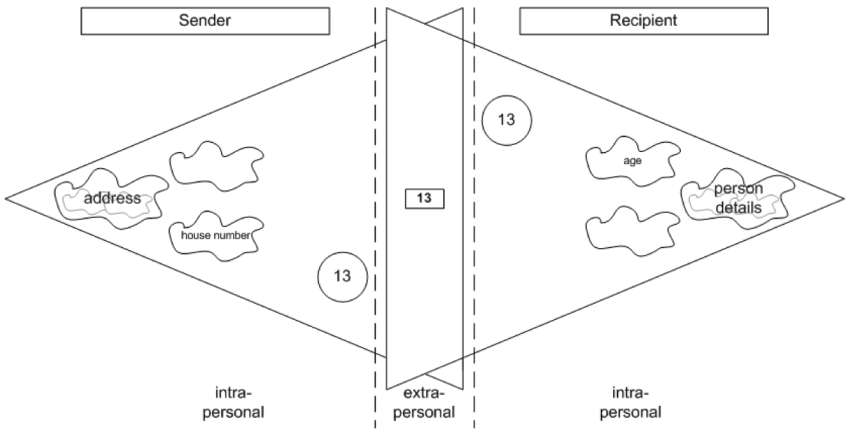


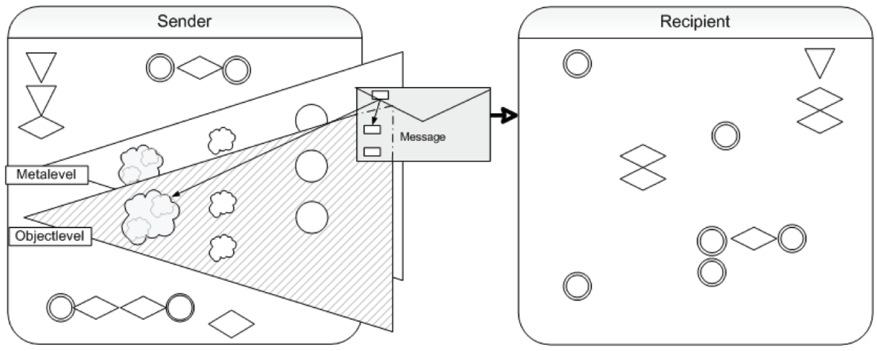
Figure 3. Sender-Recipient model for the data instance "13"



mally) known, but in written documents the future readers are unknown and their expected specific attributes are estimated by intentional group analysis. A *message* (cf. Figure 4) consists of data instances that are either on an objectlevel, describing the message itself, or at a metalevel, adding additional data to the data instances on objectlevel (such as filetype, creation date, target audience, etc.). A message is transmitted from the sender to the receiver. Both parties may have a different background and knowledge. Therefore, the message must be structured in a way that allows for an easy abstraction process. Linguistics-based modeling methods are one way to achieve such a structure of documents and will be introduced and explored in the following sections.

According to Shannon and Weaver (1964) there is always the problem of *noise* when transmitting messages (and can be extended to *semantic noise* according to Berlo (1960)). This semantic noise needs to be reduced drastically in order to allow for a faultless interpretation of the data instances.

Figure 4. Semantic communication model (Sieber & Lautenbacher, 2007)



3 LINGUISTICS-BASED MODELING METHODS

Requirements specifications are most of the times simple text files. In order to cover the intended semantics when writing them (according to the Speech Act theory), it is necessary to describe the constructs in a way in which the reader can capture the purpose of the words used by the writer. In addition it is possible to rely on existing text files of an ontology or one can start one step earlier and use existing linguistics-based modeling methods to ensure capturing the semantics into the future documents. Modeling methods are mostly used for creating structures in technical documentation and for information structuring.

Using natural language processing techniques would be another possibility. But as they are very time-consuming, we focus on the usage of modeling methods and ontologies instead. The most prominent modeling methods that are used in the context of external technical documentation for that purpose are the functional-positional segmentation method, the function design™ method, information mapping™, information structuring in XML/SGML (Lobin, 2000) and DITA. In the following we give a more detailed overview about the method of functional positional segmentation, the function design™ method as well as DITA, before we compare them.

The Functional-Positional Segmentation Method

Wiegand (1989) described the process of how a structural analysis of dictionary articles can be performed. He developed a segmentation method consisting of the method of functional segmentation and the method of functional-positional segmentation. The functional segmentation (e.g. of dictionary articles) includes the identification of functional text elements which is also interesting for modeling all kinds of documents and henceforth also for requirements specifications. The requirements specification should already exist in order to apply the functional-positional segmentation method.

Functional segmentation determines typographical and non-typographical structure pointers that assist the user in perceiving the structure of the article and of parts of an article that belong together. Typical typographical structure pointers are the font type and the font styles. Examples for non-typographical structure pointers are punctuation marks, brackets and arrows. Exhaustive functional text-segmentation is described as the incremental segmentation of a dictionary article by considering statements and structure pointers together with a presentation as whole-part relationships in a formally defined description language. This leads to a segmentation of the text in which all elements are defined and can be associated to be part of a bigger context (Figure 5).

On the other hand, the functional-positional segmentation does not only force the segmentation of the text, but also includes the position of all text segments in a linear order. The determined hierarchical structure of articles can then be presented as a tree-like partitive structure graph.

The Function Design Method

The function design™ method (Muthig & Schäflein-Armbruster, 1999) is a universal and flexible technology for structuring technical documentation. It has been developed on the basis of theoretical thoughts upon the Speech Act: each spoken sentence contains information and serves a communicative function. Nevertheless, there is no clear 1:1 correlation between a sentence and its communicative function. Most of the times the meaning of a sentence is transported using meta-spoken ways such as pronounciation as described previously. This must be achieved differently in printed documents.

Based on that, the goal of the function design™ theory is that each sentence of a text needs to have a unique identifiable function. The function design™ method can be divided into macro and micro levels: on the macro level the kind of document is classified and on the micro level this document is further segmented into sequence patterns, functional entities and tags. Therefore, the developer must put himself in the position of the recipient of a text, for example the reader of a warning. The reader of a warning needs answers such as "what is the danger?"; "how big is the danger?"; "how can I avoid the danger?"; etc.

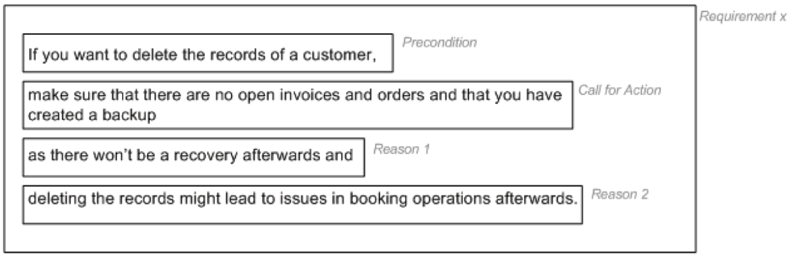
The intentions and existing knowledge of the target group must be kept in mind during the whole specification process. The created document should answer the questions a possible reader may have. Therefore, the data items in the document can be captured in functional entities such as warning, notices, preconditions etc. These functional entities must answer the questions that a potential reader might be thinking of (for example, "what do I need to consider when doing some task?"). Each functional entity is described with the following data items:

- name and (optional) abbreviation
- purpose
- sequential order
- inner structure and formulation
- layout and design

Basic functional entities can be grouped into structural patterns which have a common purpose or function. This fosters reuse of data in the function design.

The function design™ method needs to be applied while creating a new document (unlike Wiegand's method where an existing document is a precondition). The functional entities need to have the same form in the whole document. For example, a deletion action of customer records can simply be written in plain text such as "If you want to delete the records of a customer, make sure that there are no open

Figure 5. Functional-positional segmentation after Wiegand without tree structure


Requirement x
If you want to delete the records of a customer,
Precondition
make sure that there are no open invoices and orders and that you have created a backup
Call for Action
as there won't be a recovery afterwards and
Reason 1
deleting the records might lead to issues in booking operations afterwards.
Reason 2

invoices and orders and that you have created a backup as there won't be a recovery afterwards and deleting the records might lead to issues in booking operations afterwards." or it can be presented in Figure 6.

DITA

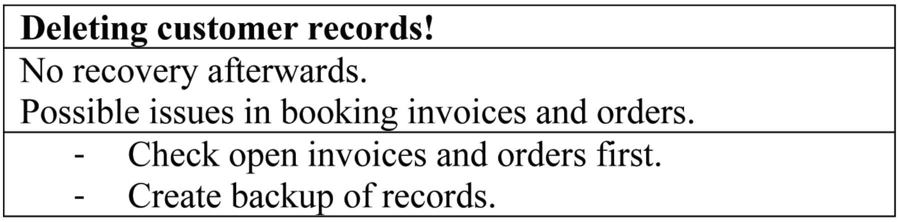
The Darwin Information Typing Architecture (DITA) (OASIS, 2007) is an information architecture based on XML especially developed for the area of technical documentation. DITA defines a set of information types that can be created and managed related to specific topics. DITA allows the authoring, producing and delivering of technical information and specifies basic elements such as topics and maps. A topic is an information entity including a title and content that is short enough to be specific to a single subject, but long enough to be self-understandable without additional references. Maps are documents that help to organize relationships to other topics.

One focus in DITA is laid on specialization (that was the reason for refering to Charles Darwin who was investigating inheritance during his studies about the theory of evolution). Inheritance and specialization allow defining new information whereas existing design is reused as much as possible. Figure 7 shows the same example as in the preceeding sections modeled in DITA.

Comparison of Different Linguistics-Based Modeling Methods

As there are some more linguistics-based modeling methods (e.g. Ament, 2003; Ley, 2007; Lobin, 2000) as described in the previous sections, we evaluated them according to several criteria. These criteria were divided into five sections: structure and documentation of the

Figure 6. Function design method example

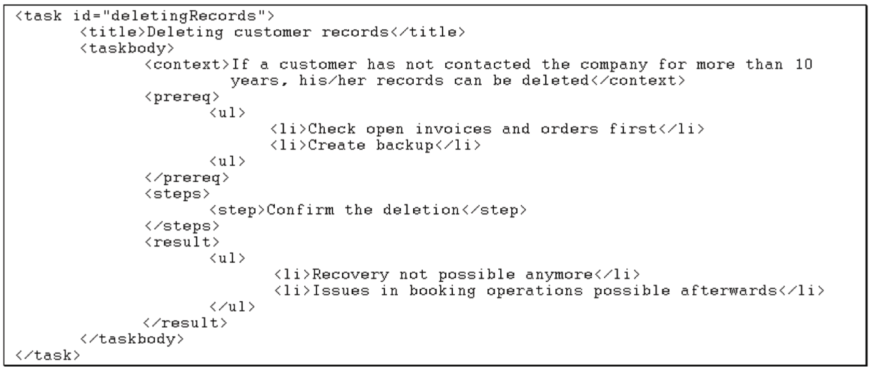


Deleting customer records!

No recovery afterwards.
Possible issues in booking invoices and orders.

- Check open invoices and orders first.
- Create backup of records.

Figure 7. Example written in DITA



```
<task id="deletingRecords">
      <title>Deleting customer records</title>
      <taskbody>
            <context>If a customer has not contacted the company for more than 10
                  years, his/her records can be deleted</context>
            <prereq>
                  <ul>
                        <li>Check open invoices and orders first</li>
                        <li>Create backup</li>
                  <ul>
            </prereq>
            <steps>
                  <step>Confirm the deletion</step>
            </steps>
            <result>
                  <ul>
                        <li>Recovery not possible anymore</li>
                        <li>Issues in booking operations possible afterwards</li>
                  </ul>
            </result>
      </taskbody>
</task>
```

modeling method, experiences when applying the modeling method to an example, evaluation of the result after applying the modeling method, how can the instances be personalized and how can they be reused in another context. The questions in the evaluation were developed similarly to other evaluation approaches such as Hevner et al. (2004), Hong et al. (1993) and Moodey & Shanks (1994). The results are only summarized here, the detailed evaluation can be found in Sieber and Lautenbacher (2007).

Concerning the comprehensibility of each modeling method, the function design™ method stands out with 37 of 40 possible evaluation points. This is probably due to the teaching of the method at universities for 10 years. DITA catches up when considering the application of the modeling method (92 of 105 possible evaluation points) as well as evaluating the quality of the results (33 of 50 evaluation points). Personalization is not really possible with any of the modeling methods (the best one had 14 of 40 evaluation points); however, DITA assists the user in reuse being nearly optimal (66 of 70 evaluation points) through a high degree of standardization as well as the use of modules that can be reused in different contexts. Concluding, DITA came first in this evaluation (with 236 of 305 possible evaluation points) ahead of function design™ (203) and information mapping (192).

4 APPLYING LINGUISTICS-BASED MODELING METHODS AND ONTOLOGIES TO REQUIREMENTS ENGINEERING

In this section we demonstrate how linguistics-based modeling methods can be used in requirements engineering and how the resulting specifications can be used as a comprehensive human-understandable basis for semantic annotation further on.

Applying DITA to an Example of a Requirements Specification

First, we introduce an example that describes requirements for a typical customer relationship management (CRM) system. In this system, personal details about customers as well as their orders are stored, created, modified and deleted. The requirements of the creation process for a new customer order (compare Metz et al., 2003) can be modeled in a textual editor (e.g. an Office product with predefined templates) as follows:

Name of scenario: Register new customer order.
Description: A new customer order is entered into the CRM system.
Context: A sales clerk with a non-processed customer order.

Main scenario:

1. The sales clerk enters the customer's ID.
2. The system displays the customer's profile.
3. The sales clerk confirms that the customer's credit rating is sufficient and the order can be processed.
4. The system assigns an order ID.
5. The sales clerk registers the desired trade items.

…

Preconditions: Customer has been stored in system, trade items are available in system.

- System has initiated an order for the customer
- System has documented payment information
- System has registered request with the customer
- System has logged all failures
- System has logged transaction date and time

- Actors involved: Sales clerk, CRM system

After the requirements have been entered in the editor, they can be stored in DITA XML-format as shown in Figure 8.

Annotating the Example by a Requirements Ontology

Linguistics-based modeling methods allow a modular access based on the defined functional entities, but they have strong limitations regarding automated processing. Therefore, it is necessary to describe the used constructs in a way machines can 'understand' (similar to Grice's description of *meaning*). That's where the Semantic Web and ontologies come into play.

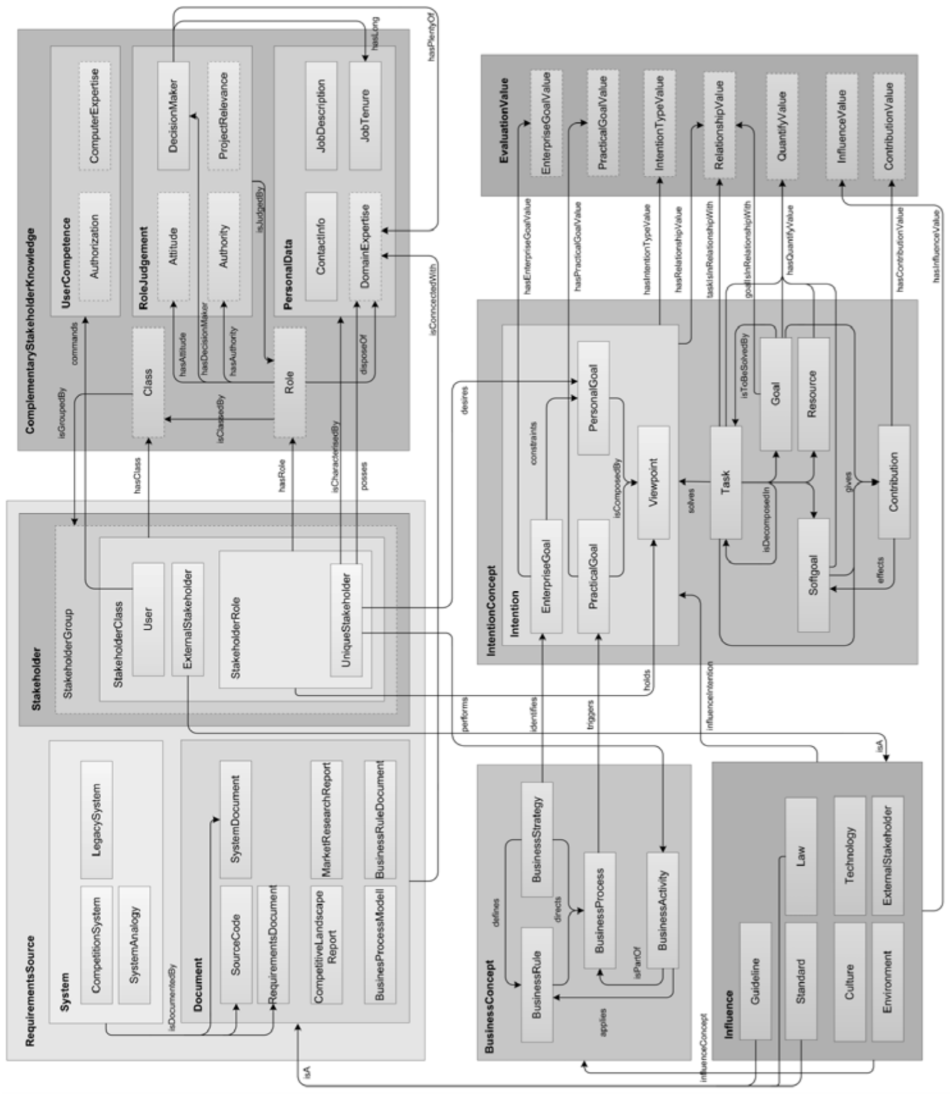
Similar to Bauer and Roser (2006) where the usage of ontologies in the context of software engineering and development is described, annotating the constructs of a requirements specification on the basis of a requirements ontology can assist the computer to process the used vocabulary. Therefore, an ontology in the Web Ontology Language (OWL; W3C, 2004) can be used and new knowledge can be gathered through the process of reasoning on this ontology. There are many approaches for annotating documents (Web pages, videos, etc.) as a means to use this data for further processing (see Euzenat, 2002; Missikoff et al., 2003). Having annotated the requirements with semantic data, this knowledge can also be used in further parts of the software engineering process as well as in document engineering.

An ontology has been defined as "a (formal) explicit specification of a (shared) conceptualization" (Gruber, 1993). There are different kinds of ontologies: Guarino (1997) differentiates between *application ontologies* that contain the definitions specific to a particular application, while *reference ontologies* refer to ontological theories whose focus is to clarify the intended meaning of terms used in specific domains. Kassal (2008) developed a reference ontology for the domain of requirements engineering that allows to capture the knowledge of all stakeholders at the beginning of a project in a formal notation. Therefore, the ontology focuses on the stakeholder (together with the complementary stakeholder knowledge), but also considers intentions, documents, business concepts or influence factors. Figure 9 shows the ontology with the entailed concepts in greater detail.

Figure 8. Customer creation requirements in DITA

```
<task id="registerNewCustomerOrder">
	<title>Register new customer order</title>
	<taskbody>
		<context>A sales clerk with a non-processed customer order</context>
		<prereq>
			<ul>
				<li>Customer has been stored in system</li>
				<li>Trade items are available in the system</li>
			</ul>
		</prereq>
		<steps>
			<step>The sales clerk enters the customer's ID</step>
			<step>The system displays the customer's profile</step>
			<step>The sales clerk confirms that the customer's credit rating
is sufficient and the order can be processed.</step>
			<step>The system assigns an order ID</step>
			<step>The sales clerk registers the desired trade items</step>
			<step>|</step>
		</steps>
		<result>
			<ul>
				<li>System has initiated an order for the customer</li>
				<li>System has documented payment information</li>
				<li>System has registered request with the customer</li>
				<li>System has logged all failures</li>
				<li>System has logged transaction date and time</li>
			</ul>
		</result>
	</taskbody>
</task>
```

Figure 9. The requirements ontology ON-EREQ (Kassal, 2008)



The requirements ontology has been instantiated with the previous example and does now contain the description of all roles and users (such as sales clerk) and the goals that were defined within a project (e.g. Register new customer order). By using this ontology (an excerpt in XML-format is shown in Figure 10) we can now annotate the customer order example in DITA.

Thereby, we use additional tags referencing parts of the ontology in analogy to current Semantic Web service standards such as SAWSDL (Kopecky et al., 2007). With these *ref*-tags, we can point from one word or a whole passage in the requirements specifications to some concepts in the ontology. This allows for improved computer processing. The result is shown in Figure 11. There, the person sales clerk is referenced to a similar named concept

Figure 10. An excerpt of the requirements ontology

```
<User rdf:about="#Sales_Clerk">
      <hasAuthorization rdf:resource="#Operator"/>
      <hasClass rdf:resource="#OperationalBusinessUser"/>
      <hasComputerExpertise rdf:resource="#Novice"/>
  </User>
  <Goal rdf:about="#Register_new_Customer_Order">
      <isToBeSolvedBy rdf:resource="#Entering_Customer_ID"/>
      <isToBeSolvedBy rdf:resource="#Register_Trade_Items"/>
  </Goal>
  <Task rdf:about="#Entering_Customer_ID">
      <isDecomposedIn rdf:resource="#Rating_Sufficient"/>
      <isDecomposedIn rdf:resource="#Customer_Profile"/>
  </Task>
```

Figure 11. Customer creation requirements in DITA with semantic annotations

```
<task id="registerNewCustomerOrder">
   <title ref="&ONEREQ;Register_new_Customer_Order">
      Register new customer order</title>
   <taskbody>
      <context ref="&ONEREQ;Sales_Clerk">A sales clerk with a non-processed
         customer order</context>
      <prereq>
         <ul>
            <li>Customer has been stored in system</li>
            <li>Trade items are available in the system</li>
         </ul>
      </prereq>
      <steps>
         <step ref="&ONEREQ;Entering_Customer_ID">The sales clerk enters the customer's
            ID</step>
         <step>The system displays the customer's profile</step>
         <step ref="&ONEREQ;Rating_Sufficient"  >The sales clerk confirms that the
            customer's credit rating is sufficient and the order can be
            processed</step>
         <step>The system assigns an order ID</step>
         <step ref="&ONEREQ;Register_Trade_Items">The sales clerk registers the desired
            trade items</step>
         <step>|</step>
      </steps>
      <result>
         <ul>
            <li>System has initiated an order for the customer</li>
            <li>System has documented payment information</li>
            <li>System has registered request with the customer</li>
            <li>System has logged all failures</li>
            <li>System has logged transaction date and time</li>
         </ul>
      </result>
   </taskbody>
</task>
```

in the ontology or the description that the rating is sufficient is connected to the concept Rating_Sufficient in the ontology.

Benefits

Using DITA it is now easily possible to derive a technical documentation for the product, since DITA has been developed exactly for this purpose. With the semantic annotations one can automatically query existing projects as to whether there had been similar use cases and how they were implemented. Looking for existing components which implement one of the mentioned requirements is also possible. Henceforth, the reuse of existing components can be further extended. A software developer does not need to know which components have been implemented in earlier projects, but using their semantic descriptions he can simply search for keywords and find the existing components and their descriptions and use them in the new project. This is possible due to the semantic annotation that has been integrated into the requirements. With the use of an inference engine the system can now compute similarities and equalities between requirements based on their described concepts. This enables one to find corresponding components and other use cases based on semantics and not only their syntax.

For example, one might search for all processes in which the identification of a customer is requested or modified. "Entering_Customer_ID" in the ontology includes the concepts "Enter" and "Customer_ID" which themselves might be inherited by "Insert" and "identification". Since "Insert" is a kind of modification and identification fits to the request, the PC can compute that this step, and hence the whole use case, must be considered.

Another benefit is the possibility to check for inconsistencies: if several requirements have been entered, they might describe different behaviors of a single system or several concepts of a domain that do not fit together. One requirement might say that every customer has exactly one address whereas the other says that the shipping address of the customer might be different than the address where the person lives or the organization is located.

Additionally, it is now possible to personalize a system to the user: since the ontology can, for example, represent the low level of computer expertise of the sales clerk, the human-machine interface can be adapted and made as simple as required.

5 RELATED WORK

The combination of requirements engineering and Semantic Web technologies was already studied in Selberg and Austin (2003). There, the Internet is described as a virtual, chaotic system which is similar to the study of requirements and the authors have shown which parts of requirements engineering could be realized with each tier of the Semantic Web layer cake. In their following report (Mayank et al., 2004) they describe how components could be semantically annotated and how this semantic annotation could be implemented. Similarly, Kinary (2003) shows how semantic annotated components could be composed to solve a problem, but there is no adoption of Semantic Web languages- Java is used instead.

Kaiya and Saeki (2005) represent a framework for the ontology-based analysis of requirements, but many technical details are missing as the report is quite high-level. Additionally, Lin et al. (1996) summarize all requirements in an ontology in order to deduce additional information and to check the consistency of the requirements. This ontology does not only cover quality of service aspects, but also organizational, structural or functional aspects. Dobson & Sawyer (2006) state that the basics for ontology-based requirements analysis has already been laid through the Requirements Modeling Language (RML) in the 1980s and how RML could be combined with current Semantic Web languages.

6 CONCLUSION AND OUTLOOK

In this article we have shown that linguistics-based modeling methods need to be considered and applied to requirements engineering. As in many other areas, multidisciplinarity is typical for this domain: in times of offshoring there are different cultures involved, there are always people with different apprenticeships (business analysts, software engineers, documentation engineers, etc.) which hinders a common understanding of the terms used. Only with the application of linguistics-based modeling methods and a semantic annotation of the terms adopted in requirements specifications these obstacles can be overcome.

The modeling methods allow for a clear structure of the document: already during writing of the requirements specifications the author is forced to think about how the content can best be presented to the future reader. Linguistics-based modeling methods are based on research on linguistics and Speech Acts which have already been used in the area of technical documentation for years. The usage of these techniques in Requirements Engineering seems therefore more than natural.

When the requirement documents are not only human-understandable, but become 'machine-understandable', then their reuse will grow intensively. Many tasks can be automated and documents for the next software engineering phase could be automatically generated with the usage of the model-driven architecture (MDA™) (Frankel, 2003). For example, semantically described requirements specifications could be used to generate semantic business process models (Hepp et al., 2005) which could then be further refined and finally executed.

The semantic annotation can also lead to further personalization of the developed software: if the person who is responsible for some requirements can be traced back, it is possible to personalize the software to the needs of each individual person as described in the requirements specification (e.g., in a developed requirements profile). By using an ontology that covers the profile of the requestor, it is possible to personalize the application to the needs of each specific user later on.

REFERENCES

W3C. (2004). *OWL Web Ontology Language Reference*. Retrieved from http://www.w3.org/TR/owl-ref/

Ament, K. (2003). *Single Sourcing: Building Modular Documentation*. Norwich, NY: William Andrew Publishing.

Austin, J. L. (1962). *How to Do Things with Words*. Cambridge, MA: Harvard University Press.

Bauer, B., & Roser, S. (2006). Semantic-enabled Software Engineering and Development. In *Proceedings of Informatik 2006* (LNI P-94, pp. 293-296). Bonner Köllen Verlag.

Berlo, D. K. (1960). *The Process of Communication*. New York: Holt, Rinehart, and Winston.

Bühler, K. (1965). *Sprachtheorie: Die Darstellungsform der Sprache*. Stuttgart, Germany: Verlang UTB.

de Cesare, S., Holland, G., Holtmann, C., & Lycett, M. (2007). Semantic-based systems development. In *OOPSLA Companion* (pp. 760).

Dobson, G., & Sawyer, P. (2006). *Revisiting Ontology-Based Requirements Engineering in the age of the Semantic Web*. Paper presented at the International Seminar Dependable Requirements Engineering of Computerised Systems at NPPs, Halden.

Euzenat, J. (2002). Eight Questions about Semantic Web Annotations. *IEEE Intelligent Systems*, *17*(2), 55–62.

Flensburg, P. (2007, August 11-14). *An enhanced communication model*. Paper presented at the 30th Information Systems Research Seminar in Scandinavia (IRIS), Tampere, Finland.

Frankel, D. S. (2003). *Model Driven Architecture – Applying MDA™ to Enterprise Computing*. New York: Wiley.

Gašević, D., Kaviani, N., & Milanović, M. (in press). Ontologies and Software Engineering. In S. Staab & R. Studer (Eds.), *Handbook on Ontologies*.

Grice, H. P. (1967). Logic and Conversation. In A. P. Martinich (Ed.), *Philosophy of Language*. New York: Oxford University Press.

Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, *5*(2), 199–220. doi:10.1006/knac.1993.1008

Guarino, N. (1997). Understanding, building and using ontologies. *International Journal of Human-Computer Studies*, *46*(2-3), 293–310. doi:10.1006/ijhc.1996.0091

Hepp, M., Leymann, F., Domingue, J., Wahler, A., & Fensel, D. (2005, October 18-20). Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In *Proceedings of IEEE ICEBE,* Beijing, China (pp. 535-540).

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105.

Hong, S., Goor, G., & Brinkkemper, S. (1993). A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies. In J. F. Nunamaker & R. H. Sprague (Eds.), *Information Systems of HICCS* (pp. 689-698).

Kaiya, H., & Saeki, M. (2005). *Ontology Based Requirements Analysis: Leightweight Semantic Processing Approach.* Paper presented at the International Conference on Quality Software (QSIC), Melbourne, Australia.

Kassal, S. (2008). *Semantic Requirements – Ontologie-basierte Modellierung von Anforderungen im Software Engineering*. Master thesis, University of Augsburg.

Kiniry, J. R. (2003). *Semantic Component Composition*. Paper presented at ECOOP, Darmstadt, Germany.

Kopecky, J., Vitvar, T., Bournez, C., & Farrell, J. (2007). SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, *11*(6), 60–67. doi:10.1109/MIC.2007.134

Ley, M. (2006). Aspekte der Informationsstrukturierung: Über Strukturierungsprinzipien, die Ebenen der Textstruktur und Dokumentgrammatiken. *Technische Kommunikation*, *28*(4), 51–53.

Lin, J., Fox, M. S., & Bilgic, T. (1996). A Requirement Ontology for Engineering Design. *Concurrent Engineering*, *4*(3), 279–291. doi:10.1177/1063293X9600400307

Lobin, H. (2000). *Informationsmodellierung in XML und SGML*. Berlin Heidelberg, Germany: Springer-Verlag.

Mayank, V., Kositsyna, N., & Austin, M. (2004). *Requirements Engineering and the Semantic Web – Part II* (Tech. Rep. 2004-14). College Park, MD: Institute for Systems Research.

Metz, P., O'Brien, J., & Weber, W. (2003). Specifying Use Case Interaction: Types of Alternative Courses. *Journal of Object Technology JOT*, *2*(2).

Missikoff, M., Schiappelli, F., & Taglino, F. (2003, October). A Controlled Language for Semantic Annotation and Interoperability in e-Business Applications. In *Proceedings of the Semantic Integration Workshop (SI-2003)*, Sanibel Island, FL (Vol. 82, 13). CEUR-WS.

Moodey, D. L., & Shanks, S. (1994). What makes a good data model? Evaluating the Quality of Entity Relationship Models. In P. Loucopoulos (Ed.), *Entity-Relationsship Approach – Proceedings of ER'94, Business Modelling and Re-Engineering* (pp. 94-111).

Muthig, J., & Schäflein-Armbruster, R. (1999). *Funktionsdesign: eine universelle und flexible Standardisierungstechnik*. Augsburg, Germany: WEKA-Verlag.

OASIS. (2007). *DITA Version 1.1, Architectural Specification*. Boston: Author.

Rueping, A. (2003). *Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects*. New York: Wiley Software Patterns Series.

Rupp, C. (2007). *Requirements Engineering und Management – Professionelle, iterative Anforderungsanalyse für die Praxis*. Munich, Germany: Hanser Verlag.

Schramm, W. (1954). How communication works. In W. Schramm (Ed.), *The Process and Effects of Mass Communication* (pp. 5-6). Urbana, IL: University of Illinois Press.

Searle, J. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge, UK: Cambridge University Press.

Selberg, S. A., & Austin, M. (2003). *Requirements Engineering and the Semantic Web – Part I*, (Tech. Rep. 2003-20). College Park, MD: Institute for Systems Research.

Shannon, C. F., & Weaver, W. (1964). *The Mathematical Theory of Communication*. Urbana, IL: University of Illinois Press.

Sieber, T., & Kammerer, M. (2006). Sind Metadaten bessere Daten? *Technische Dokumentation*, *5/2006*, 56–58.

Sieber, T., & Kovács, L. (2005). Technical documentation: Terms, problems and challenges in managing data, information and knowledge. In Proceedings of the *University of Miskolc's 5th International Conference of PhD Students* (pp. 165-170).

Sieber, T., & Lautenbacher, F. (2007). *Enterprise Content Integration: Documentation, Implementation and Syndication using Intelligent Metadata (ECI-DISI)* (Tech. Rep. 2007-17). Augsburg, Germany: University of Augsburg, Germany. Retrieved from http://www.ds-lab.org/publications/reports/2007-17.html

Strawson, P. F. (1969). Intention and Convention in Speech Acts. In K. T. Fann (Ed.), *Symposium on J. L. Austin, International Library of Philosophy and Scientific Method*. New York: Humanities Press.

Tetlow, P., Pan, J. Z., Oberle, D., Wallace, E., Uschold, M., & Kendall, E. (2006). *Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering*. Cambridge, MA: W3C.

Wiegand, H. E. (1989). Aspekte der Makrostruktur im allgemeinen einsprachigen Wörterbuch: alphabetische Anordnungsformen und ihre Probleme. In F. J. Hausmann, O. Reichmann, H. E. Wiegand, & L. Zgusta (Eds.), *Wörterbücher* (Vol. 1, pp. 371-409). Berlin, Germany: Springer.

Wiegand, H. E. (1989). Der Begriff der Mikrostruktur: Geschichte, Probleme, Perspektiven. In F. J. Hausmann, O. Reichmann, H. E. Wiegand, & L. Zgusta (Eds.), *Wörterbücher* (Vol. 1, pp. 409-462). Berlin, Germany: Springer.

Wittgenstein, L. (1973). *Philosophical investigations*. Upper Saddle River, NJ: Prentice Hall.

Since 2005 Florian Lautenbacher is researcher and Ph.D. student at the University of Augsburg, Germany, and holds a diploma in Computer Science from the same university. During his studies he has worked for Fujitsu Siemens Computers, a medical supply center as well as at the university. His current research interests are in applying semantic technologies to model-driven software engineering, in particular in workflow and business process technologies as well as service-oriented architectures. Florian is project co-lead of the Eclipse Technology project Java Workflow Tooling (JWT) which started in 2007 and is part of the current Eclipse Galileo distribution. Moreover he is involved in several other national and international projects mostly related to business process management and SOA. He has published more than 20 scientific papers.

Bernhard Bauer is professor and head of the programming of Distributed Systems Group at the University of Augsburg since 2003. He holds a diploma in computer science from the University of Passau, Germany, and a PhD. in computer science from the Technische Universität München, Germany. For more than 6 years Bauer has worked in industry. The focus of his research group at the university is on industrialization of software engineering and software operation. The main research areas are model-driven software development, semantic technologies as well as self-organizing systems to improve the automation of the software product lifecycle as well as the autonomy of software systems. He has published more than 100 scientific papers in the area of agent-based systems and agent-oriented software engineering, compiler construction, (semantic-enabled) model-driven software engineering and autonomous systems.