GINO - A Guided Input Natural Language Ontology Editor

Abraham Bernstein and Esther Kaufmann

University of Zurich, Dynamic and Distributed Information Systems, Switzerland
{bernstein,kaufmann}@ifi.unizh.ch

Abstract. The casual user is typically overwhelmed by the formal logic of the Semantic Web. The gap between the end user and the logic-based scaffolding has to be bridged if the Semantic Web's capabilities are to be utilized by the general public. This paper proposes that controlled natural languages offer one way to bridge the gap. We introduce GINO, a *guided input natural language ontology editor* that allows users to edit and query ontologies in a language akin to English. It uses a small static grammar, which it dynamically extends with elements from the loaded ontologies. The usability evaluation shows that GINO is well-suited for novice users when editing ontologies. We believe that the use of guided entry overcomes the *habitability problem*, which adversely affects most natural language systems. Additionally, the approach's dynamic grammar generation allows for easy adaptation to new ontologies.

1 Introduction

The Semantic Web's logical underpinning provides a stable scaffolding for machine-based processing. The common or occasional user, however, is typically overwhelmed with formal logic. The resulting gap between the logical underpinning of the Semantic Web and the average users' ability to command formal logic manifests itself in at least two situations. First, the gap manifests itself when the untrained user tries to use an existing, usually graph-based, ontology editing tool [14, 12] – the *editing disconnection*. Second, it can be found in the disconnection between a user's information needs and the query (language) with which the user tries to find the required information in an ontology [28, 27, 9] – the *querying disconnection*. Since editing and querying are two of the major interaction modes with the Semantic Web, bridging them is central to its practical use by end users. Consequently, the question how to bridge the gap is pivotal for the success of the Semantic Web for end users. This paper proposes to address these two manifestations of the gap using natural language interfaces (NLIs).

NLI systems have the potential to bridge the editing disconnection between the untrained user and the triple- and graph-based ontology editing/creating tools. Although there are good ontology building tools [10, 17, 22, 3, 29, 30] editing and building ontologies is hard for experts but close to impossible for common and occasional users [26]. NLIs can help to overcome this gap by allowing users

to formulate their knowledge domain and information needs in familiar natural language (NL), rather than having to learn unfamiliar formal and complex data manipulation and query languages. The major drawback of NLIs, however, is their adaptivity to new domains. Even though natural language processing (NLP) has made good progress in recent years, much current NLI research relies on techniques that remind users more of information retrieval than NLP [19]. The systems that can perform complex semantic interpretation and inference tend to require large amounts of domain-specific knowledge- and engineering-intensive algorithms making the systems hardly (if any) adaptable to other domains and applications. Hence, they have a substantial *adaptivity barrier*.

Even if we could provide domain-independent NLI a second problem would arise from the users' side. Typically, users do not know what capabilities a NL system has. Therefore, many of their assertions/questions will not be understood correctly or might even be rejected because the statements exceed or fall short of the capability of the system. The mismatch between the users' expectations and the capabilities of a NL system is called the *habitability problem* [32]. Thus, for the successful use of NLI, users need to know what is possible to say/ask [2]. Analogously, NLI can help addressing the querying disconnection assuming that the adaptivity barrier and the habitability problem can be overcome. As a consequence, the domain-dependency of intelligent NLI and the habitability problem account for the fact that we are still far away from the successful use of full NL to command and query the Semantic Web (and arbitrary information systems). In this paper, we argue that we can address the before-mentioned problems by using a *guided and controlled NLI* that supports the user in both the tasks of ontology building and query formulation. We present GINO, the **g**uided **i**nput **n**atural language **o**ntology editor for the Semantic Web. GINO, an extension of the purely querying focused Ginseng [5], essentially provides quasi-NL querying and editing access to any OWL knowledge base [18]. It relies on a simple static sentence structure grammar which is dynamically extended based on the structure and vocabulary of the loaded ontologies. The extended grammar can be used to parse sentences, which strongly resemble plain English. When the user enters a sentence, an incremental parser relies on the grammar to constantly check the user's entries to (1) propose possible continuations of the sentence similar to completion suggestions in Unix shells or "code assist" (or intellisense) in integrated development environments and (2) prevent entries that would not be grammatical and, hence, not executable/interpretable. Once a sentence is fully entered, GINO uses some additional statement construction information in the grammar to translate the quasi English sentence into new triple sets (to add/change the ontology) or SPARQL statements [25] and pass them on to the Semantic Web framework Jena for execution.

The main difference between GINO and full NLIs [2] is that GINO does not use any predefined lexicon beyond the vocabulary that is defined in the static sentence structure grammar and provided by the loaded ontologies. Furthermore, it does not try to semantically understand the entries. Instead, GINO "only knows" the vocabulary that is being defined by the grammar and by the currently

loaded ontologies. It relies directly on the semantic relationships of the loaded ontology. Hence, the vocabulary is closed and the user has to follow it limiting the user but ensuring that all queries and sentences "make sense" in the context of the loaded ontologies and can be interpreted by simple transformations.

The remainder of the paper is structured as follows. First, we will introduce GINO by describing how the user experiences GINO as an ontology building and editing tool. Next, we will provide an overview of its technical setup and functionality. We will then describe the empirical evaluation of the approach and discuss the results, which leads to a discussion of GINO's limitations. The paper closes with a section on related work and some conclusions.

2 GINO - The User Experience

GINO allows users to query any OWL knowledge base using a guided input NL akin to English. The user enters the query or sentence in English into a free form entry field (as shown in Fig. 1). Based on the grammar, the system's incremental parser offers the possible completions of the user's entry by presenting the user with choice pop-up boxes. These pop-up menus offer suggestions on how to complete a current word or what the next word might be. Obviously, the possible choices get reduced as the user continues typing. Fig. 1 shows that typing the letter "c" within the middle of a query or sentence causes the interface to propose all the possible completions of the words that begin with "c."

Fig. 1. The GINO user interface

Users can navigate the pop-up with the arrow keys or with the mouse and choose a highlighted proposed option with the space key. Entries that are not in the pop-up list are ungrammatical and not accepted by the system. In this way, GINO guides the user through the set of possible sentences preventing statements unacceptable by the grammar. Once a sentence is completed, GINO translates the entry to triple-sets or SPARQL statements. Users who are familiar with the common graph representations or with other ontology editors basing on graph structures (e.g., Protégé [22]) can also edit elements of an ontology by using the graph structure on the right side of the interface window (in Fig. 1).

Consider a user who wants to construct a class *lake*, a datatype property *lakeDepth*, and an instance *tahoe* to which the value of the before specified property is added. To create a new class *lake* and add it to an ontology that contains geographical information the user starts typing a sentence beginning with "there is" or "there exits." The pop-up shows possible completions of the sentence and the user can continue the sentence with "a" and "class." Choosing "class" (as we want to create a class) leads to the alternatives "named" and "called." Either choice then prompts the user to enter the class's label (i.e., "lake"). Finishing the sentence with a full stop prompts GINO to translate the completed sentence "There is a class named lake" into corresponding OWL triples that are loaded into the Jena ontology model, thereby enabling that the class can be queried or offered in a pop-up. To ensure consistency all entries are then checked by the JENA Eyeball RDF/OWL model checker (http://jena.sourceforge.net/Eyeball/). The newly produced class is immediately displayed in the graph representation on the right side of the user interface.

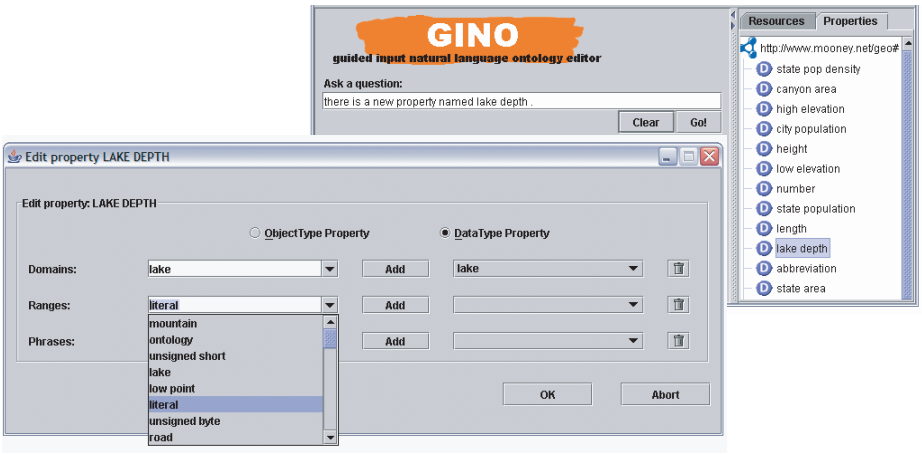

Fig. 2. The GINO user interface and property editing window

In order to specify a datatype property *lakeDepth* to the class *lake*, the user again starts a sentence with "there is." After entering "a" or "a new", the user is offered "property" to choose. Next, the label of the property has to be defined. When finishing the entry "There is a new property named lake depth" with a full stop, a window opens where the user can now specify whether the new property is a datatype or an object property (Fig. 2). Furthermore, domain and range can be specified. GINO offers the possible choices for the domain/range specification by showing the existing classes and datatypes in a pop-up. The user can, for example, choose the previously created class *lake* as domain of the property and click on "add" to actually add the chosen class. If the property has been declared as datatype property, GINO offers "literal" as possible entry for the range of the

property. Again, the user adds the range to the property *lakeDepth*. Clicking on "ok" closes the property editing window and adds the specified information to the Jena ontology model. The new property appears in the graph representation as datatype property (Fig. 2). Object properties are created analogously.

An instance of the class *lake* can now be added by entering a simple sentence beginning with "there is an instance." After continuing the sentence with "of" and "class" GINO's popup offers the list of currently defined classes. Having chosen "lake" the user can then add a label (e.g., *tahoe*) to the new instance analogously to when entering a new class resulting in "there is an instance of class lake named tahoe." Alternatively, the user could have entered a sentence "there is a lake named tahoe" where GINO would have listed the possible classes at the position of "lake' in the sentence. Values of instance attributes can also be entered using a NL input sentence, e.g. "the depth of lake tahoe is 1645 feet."

The graph representation on the right side of the GINO user interface offers an overview of the classes, properties, and instances as well as an easy editing function. By double-clicking on an element, an edit window is opened where the user can add, change, or delete elements, values, etc. Double-clicking on the instance "tahoe" in the instance tree, for example, opens an edit window showing the possible properties of the class to which the instance belongs. The value "1645" can be entered as literal of the property *lakeDepth*.

3 GINO's Technical Design

From an architectural point of view, GINO has four parts (see Fig. 3): a grammar compiler, a partially dynamically generated multi-level grammar, an incremental parser, and an ontology-access layer (i.e., Jena; http://jena.sourceforge.net/). When starting GINO, all ontologies in a predefined search path are loaded. For each ontology, the *grammar compiler* generates the necessary dynamic grammar rules to extend the static part of the grammar, which contains the ontology-independent rules specifying general sentence structures. The *grammar* is used by the incremental parser in two ways: First, it specifies the complete set of parsable questions/sentences, which is used to provide the user with alternatives during entry and prevent incorrect entries as described above. Second, the grammar also contains information on how to construct the SPARQL statements from entered sentences. Thus, a complete parse tree of an entered question can be used to generate the resulting SPARQL statements to be executed with Jena's SPARQL engine ARQ. As SPARQL does not offer any data manipulation statements (e.g., CREATE, INSERT, DELETE) we have to specify and execute the generation and insertion of the corresponding triples separately by using the Jena API in GINO's source-code.

The *incremental parser* maintains an in-memory structure representing all possible parse paths of the currently entered sequence of characters. This has various benefits. First, it allows the parser to generate a set of possible continuations (i.e., possible next character sequences by expanding all existing parse paths, which are displayed by GINO's popup). One parse path might generate

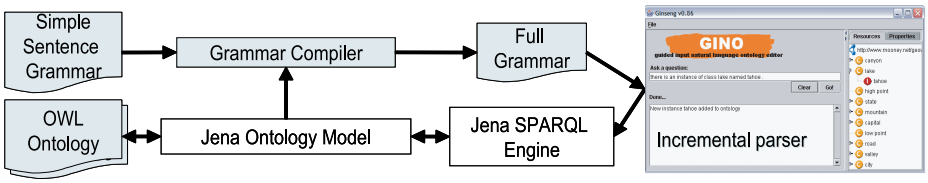

Fig. 3. The GINO architecture

multiple options when the parser expands a non-terminal being specified in more than one place in the grammar. Second, the parser can compare every character entered against the possible entries providing immediate feedback when the user attempts to enter an non-interpretable (i.e., non-parsable) sentence/character to mitigate the habitability problem. Third, when the user has finished entering the sentence, the parser can immediately provide the set of acceptable parse paths. When querying a simple transformation relying on the query construction grammar elements can translate the parse paths to SPARQL queries avoiding lengthy semantic interpretation (and possible delays in answering the query) of the sentence as usual in NLIs. The fact that there might be multiple parse paths possibly being ambiguous is simply handled by returning the union of answers back to the user. When making assertions, GINO could use the parse paths to alleviate possible ambiguities by asking the user. Currently, however, we assured that the assertions grammar is unambiguous not requiring this interaction.

Since both the use of an ontology access layer and the construction of an incremental parser are well documented in the literature [5], the rest of the section will focus on the functionality of the grammar and the grammar compiler.

3.1 The Functionality of the Grammar

The grammar describes the parse rules for the sentences that are entered by the user. Consider the following grammar excerpt as an example:

```
(1) <START> ::= there is a <NS_S> .
(2) <NS_S> ::= class named <ENTER_NEW_CLASS_NAME> .
(3) <NS_S> ::= <CLASS> named <ENTER_NEW_INSTANCE_NAME> .
(4) <CLASS> ::= <NCc>|<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <NCc>
(5) <NCc> ::= water area|<http://www.mooney.net/geo#state>
(6) <ENTER_NEW_CLASS_NAME> ::= enter_new_class_name
(7) <ENTER_NEW_INSTANCE_NAME> ::= enter_new_instance_name
```

The grammar's representation mostly follows the Backus-Naur-Form notation: Non-terminal symbols use uppercase characters (e.g., `<CLASS>`), whereas terminal symbols such as `named` that can be displayed to the user in a pop-up use lowercase characters. Grammar elements after the "|" symbol denote type restrictions. Note that we have radically simplified the example rules to keep things understandable.

While parsing, the incremental parser recursively searches for possible matches for the symbols on the left side of the rules and replaces them by the symbols

on the right side of a conformable rule. The parse is completed when no non-terminal symbols are left. By keeping every replacement step during the parsing process, a parse tree of an entered sentence is successively built. Every sentence starts with the `<START>` symbol. To replace the `<START>` symbol, this simple grammar offers the terminal symbols `there is a` followed by the non-terminal symbol `<NS_S>` and a full stop (rule 1). The terminal symbol is displayed to the user in a pop-up menu as possible beginning of an entry. If the user enters "there is a", then the parser can bind `<START>` to rule 1. Next, the parser tries to match the non-terminal `<NS_S>`, for which this grammar offers two rules (2 and 3). If the user enters "class named", the parser binds rule 2 to `<NS_S>` and discards rule 3. To replace the symbol `<ENTER_NEW_CLASS_NAME>` in rule 2, only rule 6 can be applied. The application of rule 6 replaces the non-terminal symbol by the terminal symbol `enter_new_class_name`. This special terminal symbol additionally causes the interface to provide the user with a text entry field. If the user enters a string as label of the class (e.g., *lake*) and finishes the sentence with a full stop, GINO uses the complete parse tree to generate the appropriate OWL triples and loads them into the Jena ontology model. The new class is shown in the graph representation of the ontology. After entering "there is a" according to rule 1, rules 3 and 4 provide the list of all possible class-labels. In our limited grammar, only rule 5 binds to the terminal <`CLASS`>. As an alternative to entering "class named" as described above the user can, thus, also choose (one of) the class labels that are shown to the user (e.g.,*water area*).

3.2 The Grammar Compiler

When loading an ontology, GINO generates a dynamic grammar rule for every class, property, and instance. These dynamic rules enable the display of the labels used in the ontology in the pop-up boxes. While the *static grammar rules* (all rules above except rule 5) provide the basic sentence structures, the *dynamic rules* (rule 5) allow that certain non-terminal symbols of the static rules can be "filled" with terminal symbols (i.e., the labels) that are extracted from the ontology model or provide the structure to specify relationships between elements in the ontology.

The *static grammar rules* provide the basic syntactic structures and phrases for questions and declarative sentences. Its rules supply a small set of declarative sentence structures such as "`There is a subclass of class` water area `named` lake`.`" (static grammar terminals in courier) in order to ensure the correct translation into OWL syntax. The same grammar also handles general question structures as "`Which` state borders Georgia`?`" as well as other types of queries such as closed questions ("`Is there a` city `that is the` highest point `of a` state`?`", typically resulting in an answer of "yes" or "no") or questions resulting in numbers (e.g., "`How many` rivers run through Georgia`?`"). Furthermore, it provides sentence construction rules for the conjunction or disjunction of two phrases (or sentence parts). The static grammar consists of about 120 mostly empirically constructed domain-independent rules. We are currently working on specifying these rules in an OWL-relying syntax such as SWRL (http://www.daml.org/2003/11/swrl/) to

be able to use consistency checking and other features from standard Semantic Web APIs.

The *dynamic grammar rules* get generated from the loaded OWL ontologies (rule 5 in the above grammar example). The grammar compiler essentially parses an ontology and generates a rule for each class, instance, object property, and data type property. To illustrate the dynamic rule generation, we will show the translation of an OWL class into its corresponding generated rules. Consider the OWL class definition (in the file specifying the URIs at http://www-mooney.net/geo):

```
<owl:Class rdf:ID="waterArea">
   <gino:phrase rdf:value="water areas"/>
</owl:Class>
```

Its transcription generates two GINO rules for noun clauses; one for the actual class definition and one for the `gino` tag facilitating that plurals of nouns can be used in GINO. Since both labels start with a consonant, the resulting rules are describing the non-terminal `<NCc>` for **n**oun **c**lause **c**onsonants as follows (rather than `<NCv>` for **v**owels[1]):

```
<NCc> ::= water area|<http://www.mooney.net/geo#waterArea>
<NCc> ::= water areas|<http://www.mooney.net/geo#waterArea>
```

GINO also allows that synonyms of the labels used in the ontology model can be included by annotating the ontology with additional tags from the `gino` name space. As such, GINO generates a dynamic grammar rule for each synonym.

```
<owl:Class rdf:ID="waterArea">
   <gino:phrase rdf:value="water areas"/>
   <gino:phrase rdf:value="body of water"/>
   <gino:phrase rdf:value="bodies of water"/>
</owl:Class>
```

While such annotations are not necessary for GINO to run correctly, they do extend its vocabulary and increase its usability. Additionally, they reduce the limitation that the approach, to some extent, depends on the choice of vocabulary, when the ontology was built. The more meaningful the labels of an ontology are chosen, the wider and more useful the vocabulary provided by GINO is.

4 Usability Evaluation

To get a first feedback on the usability of GINO, we confronted six users who had no experience in ontology building and editing whatsoever with our prototype written in Java. We intended to find out how the controlled NLI of GINO can support untrained and casual users in an ontology creating and editing task and help overcome the editing disconnection. Note that we did not test GINO's ability to overcome the querying disconnection, as it has already been addressed in the literature (see the related work section for more details [5, 13, 23, 6, 32]).

[1] Thus, we handle determiner-noun (e.g., *a class* vs. *an instance*) and also subject-predicate (e.g., *Which class is...* vs. *Which classes are...*) agreement.

4.1 Setup of the Experiment

The experiment was based on the *Mooney Natural Language Learning Data* [31]. Its geography database consists of a knowledge base that contains geographical information about the US and their logical representations. To make the knowledge base accessible to GINO we translated it to OWL and designed a simple class structure as meta model. We removed the class *lake* and the class *river* including their instances in order to make the experiment realistic. We recruited six subjects who were not familiar with Semantic Web technologies and ontologies; they did not even know what an ontology was in the sense of the Semantic Web. We purposely recruited people with no computer science background as GINO is intended for casual or occasional users. Each subject was given a two-page introduction on what the idea of the Semantic Web is and how contents of ontologies are basically specified (i.e., subject - predicate - object). The subjects were first asked to enter a query into GINO in order to get used to the tool. We then gave the subjects the following tasks with respect to the adapted geography ontology. The subjects were first asked to create a class *waterArea*. Second, they had to specify a new class *lake* as a subclass of *waterArea*. Next, the subjects had to define a datatype property *lakeDepth* as well as an object property *isLocatedIn* with the domain *lake*. They were then requested to add an instance *tahoe* to the class *lake* and to enter values for the two properties' ranges that they had defined before. Finally, the subjects had to change the value for the depth of Lake Tahoe (*lakeDepth*) from metric to the English units.

Using a key-logger, we logged and timed each key entry. At the end of the experiment, we performed the SUS standardized usability test [7] – a standardized collection of questions (e.g., "I think that the interface was easy to use.") each answered on the Likert scale providing a global view of subjective assessments of usability. The test covers a variety of usability aspects such as the need for support, training, and complexity. To collect more specific details on how the subjects experienced GINO we followed up on each SUS question with either "If you disagreed, what did you find difficult?" or "If you agreed, what did you find especially easy?"

4.2 Results of the Experiment

To our surprise all subjects successfully performed the given tasks having only minor difficulties with the user interface (such as clicking on the wrong button). Each subject managed to correctly add the two classes, two properties of different type, and an instance including the specification of the values for its properties. As only "mistake" three of the subjects mixed up the definition of domain and range when entering the object property, but immediately corrected their error after reconsulting the instructions. One subject even wrote down that the domain corresponded to *subject* and the range to *object* of the subject-predicate-object triple structure. This questions the suitability of these mathematics-rooted terms for casual users. Examining the entry logs we found that the subjects corrected very few of their entries (e.g., using the backspace button) indicating that they

had quickly learned the capabilities of GINO's NL parser. Even though we can only hypothesize that this was due to GINO's popup-based guidance, we know that other NLIs without that feature suffer from the habitability problem.

The users gave GINO an average SUS score of 70.83 (σ = 11.15, median = 73.75), which ranges from 0 to 100. As usability is not an absolute criterion the resulting score can only be understood relatively to others [21, 24]. A similar SUS evaluation of two NLI-based query systems, for example, resulted in average SUS scores of 49.29 for GINO's predecessor Ginseng [5] and 52.14 for a controlled English based NLI SWAT [6]. As the SUS score shows, users found GINO significantly better suited to the task. This is a very good result considering that our subjects were *unfamiliar* with ontology issues before the experiment and that "they were thrown in the deep end of ontology building/editing tasks" after a very general and brief introduction.

To the questions "Why would you like to use GINO frequently?" and "Why would most people learn to use GINO quickly?" the answer "easy and intuitive" was given four out of six times. Two subjects thought that the interface had a clear and logic design. Nevertheless, four subjects reported that they would prefer the support of a person for the first time, but after that they would be able to use GINO on their own. One person specifically stated that the most convenient feature of GINO was that one can enter elements using NL. Five subjects found the Semantic Web fundamentals were the most difficult and time-consuming part of the experiment. The subjects did not use the right-hand graph view to do any editing apart from the task where they were explicitly asked to do so.

4.3 Discussion and Summary of Experimental Results

Obviously, our test does not provide (final) quantitative proof of GINO's suitability for ontology editing by casual users. As we will discuss in the limitations section below, we need a full user evaluation with many subjects from a variety of backgrounds to provide such evidence. Nevertheless, our experiment strongly indicates that novice users (1) can edit ontologies with a NL-based ontology editor without being overwhelmed by formal logic, (2) can do so with virtually flawless results, (3) preferred using the NL entry to a direct manipulation graph-based view, and (4) provided some interesting insight into the confusion behind the semantics of the terms the Semantic Web uses for the general public. Consequently, we can conclude that the GINO NLI has great potential to overcome the gap between the average user's ability to command formal logic and the Semantic Web's logic-based scaffolding. It also seemed to successfully circumvent the habitability problem.

5 Limitations and Future Work

Although the preliminary usability evaluation showed promising results, the approach has its limitations. First, the evaluation is limited with regard to the

number and choice of the subject pool as well as the extent of the task. Furthermore, our subjects had never used any other ontology tool before the experiment, therefore making a comparison with these tools impossible. To address this issue we intend to undertake more intensive user testing in the future to determine whether NLIs are capable of bridging the logic gap. Such experimentation would include an extended editing task, a larger subject pool, and the use of additional tools as a benchmark. Specifically, we need to compare GINO with a simplified version of an ontology editor (such as Protégé [22]) to establish whether the users' ability to edit the ontology came from the *deliberate simplification* of GINO or from its *langauge capabilities*. Nonetheless, we think that our results are extremely encouraging and provide a strong indication that GINO enabled the casual/novice users to correctly accomplish a simple ontology editing task. Users who intend to embark on large ontology editing/design tasks, however, might prefer to invest the time to learn a full-fledged ontology editor.

Second, GINO is not a full-fledged ontology building and editing tool. It deliberately has a simple design to allow its use by novices (who might be overwhelmed by advanced logic features such as quantified restrictions). However, the simple approach can also be regarded as a strength, since the casual user is able to handle the tool and is not confused by many complex functions.

Furthermore, the controlled language limits the expressiveness of the user, but this restriction is not overly severe as shown in the evaluation. We think that the limitation is justified by two benefits. First, by using a controlled NL, we can avoid one of the biggest problems of NL: ambiguity. Handling ambiguity, in turn, is still regarded as a prerequisite for the successful usage of NLIs [20]. Second, the use of a controlled language (together with a guidance feature) addresses the habitability problem.

One question which we left unanswered was GINO's scalability. As the grammar compiler generates at least one rule for every class/instance/property in the ontology, the grammar is likely to grow very fast for large ontologies. We believe that this issue could be easily ddressed by using standard knowledge-base optimization techniques such as storing instances related rules on disk and retrieving them only when needed.

In the future, we intend to specifically address the adaptivity barrier. To adapt a new ontology of the size of the geography ontology to GINO took us about one hour. The task mainly consisted of adding synonyms of words and word phrases as `gino` tags to the ontology. Given a graphical user interface this task could be accomplished by a novice. M-PIRO [1], a tool for multi-lingual ontology-based language generation, for example, found that users could do so easily. Note that such a tool could be extended with user support functions based on WordNet (http://wordnet.princeton.edu/).

6 Related Work

The idea of NLIs is not new. NLIs to databases have repeatedly been developed since the seventies, but oftentimes with moderate success [2, 9, 19, 32]. Consider-

ing the difficulties with full NL, the side step to restricted NL or menu-guided interfaces, which has often been proposed, seems obvious [8, 4, 11, 32]. Even though there exist good and sophisticated ontology construction and editing tools (e.g., Ontolingua [10], Chimaera [17], OilED [3], Protégé [22], OntoEdit [29]), they all follow the menu/graph-based user interface paradigm. Swoop [12] tries to make use of people's familiarity with standard web browsers and offers a user interface that reflects the "webiness" with which people are used to interact.

There are some NLI-based projects that closely relate to GINO. [26] show how (OWL) ontologies can be constructed using a controlled NL to express the specifications. Their approach relies on a bidirectional grammar that translates entered facts, axioms, and restrictions into OWL abstract syntax. The controlled language does not have to be learned as a text editor guides the user through the writing process by offering look-ahead information (i.e., word categories). Unfortunately, no user evaluation is reported on. One major difference between their approach and GINO is that their lookahead feature does not show the actual words but displays the acceptable word-categories. This enhances the cognitive load on the user, as he/she has to map the grammatical word-category to possible words. Also, their project is aimed at finding an alternative notation to OWL resulting in logic-alike statements such as "*Iff X is a pizza then X is a dish.*" While this might be grammatically correct English, it is clearly not aimed at the casual user but at someone who understands the general principles of first-order logic. In contrast, [30] offers a simple controlled language for specifying ontologies. However, it does not provide direct guidance or look-ahead support.

LingoLogic [32] is a user interface technology that combats the habitability problem by using menus to specify NL queries and commands that can be executed on relational databases. A parser checks a user's entries, displays possible completions of the words/phrases to the user, and translates the entries to the target query language SQL. In contrast to GINO LingoLogic seems to be limited to querying databases. We did not find any data manipulation capabilities. Hence, LingoLogic seems to belong to the category of NLI-based ontology/database querying tools such as PRECISE [23], AquaLog/PowerAqua [16, 15], START [13], SWAT [6], or Ginseng [5] – the GINO predecessor. PRECISE, Ginseng, and SWAT were evaluated with the Mooney databases [31], which also have associated NL queries collected from users. Ginseng and SWAT were additionally tested with end users in a task setting. All evaluations show that NLIs can be successfully used to overcome the querying disconnection. As such, they complement our findings, which focused on bridging the editing disconnection.

7 Conclusions

In order to be usable to the casual or novice user, the logic-based scaffolding of the Semantic Web needs to be made accessible for editing and querying. We propose that NLIs offer one way to achieve that goal. To that end, we introduced GINO, the guided input natural language ontology editor that allows users to edit and query ontologies in a quasi-English guided language. Our evaluation

with six end users provides some evidence that novice users are capable of virtually flawlessly add new elements to an ontology. It also showed that end users were confused by the terms for the major Semantic Web elements that the research community currently uses. Additionally, we found that the use of guided entry seemed to overcome the habitability problem that hampers users' ability to use most full NLIs. We believe that this paper shows the potential of NLIs for end user access to the Semantic Web, providing a chance to offer the Semantic Web's capabilities to the general public.[2]

References

1. I. Androutsopoulos, S. Kallonis, and V. Karkaletsis. Exploiting owl ontologies in the multilingual generation of object. In *10th European Workshop on Natural Language Generation (ENLG 2005)*, pages 150–155, Aberdeen, UK, 2005.
2. I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *Natural Language Engineering*, 1(1):29–81, 1995.
3. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. Oiled: A reason-able ontology editor for the semantic web. In *Intl. Description Logics Workshop*, Stanford, CA, 2001.
4. S. Bechhofer, R. Stevens, G. Ng, A. Jacoby, and C. Goble. Guiding the user: An ontology driven interface. In *1999 User Interfaces to Data Intensive Systems (UIDIS 1999)*, pages 158–161, Edinburgh, Scotland, 1999.
5. A. Bernstein and E. Kaufmann. Making the semantic web accessible to the casual user: Empirical evidence on the usefulness of semiformal query languages. *IEEE Transactions on Knowlwdge and Data Engineering*, under review.
6. A. Bernstein, E. Kaufmann, A. Göhring, and C. Kiefer. Querying ontologies: A controlled english interface for end-users. In *4th Intl. Semantic Web Conf. (ISWC 2005)*, pages 112–126, 2005.
7. J. Brooke. Sus - a "quick and dirty usability scale. In P. Jordan, B. Thomas, B. Weerdmeester, and A. McClelland, editors, *Usability Evaluation in Industry*. Taylor Francis, London, 1996.
8. S. K. Cha. Kaleidoscope: A cooperative menu-guided query interface (sql version). *IEEE Transactions on Knowledge and Data Engineering*, 3(1):42–47, 1991.
9. S. Chakrabarti. Breaking through the syntax barrier: Searching with entities and relations. In *15th European Conf. on Machine Learning (ECML 2004)*, pages 9–16, Pisa, Italy, 2004.
10. R. Fikes, A. Farquhar, and J. Rice. Tools for assembling modular ontologies in ontolingua. In *AAAI/IAAI*, pages 436–441, 1997.
11. C. Hallett, R. Power, and D. Scott. Intuitive querying of e-health data repositories. In *UK E-Science All-hands Meeting*, Nottingham, UK, 2005.
12. A. Kalyanpur, B. Parsia, and J. Hendler. A tool for working with web ontologies. *Intl. Journal on Semantic Web and Information Systems*, 1(1):36–49, 2005.
13. B. Katz, J. Lin, and D. Quan. Natural language annotations for the semantic web. In *Intl. Conf. on Ontologies, Databases, and Applications of Semantics (ODBASE 2002)*, Irvine, CA, 2002.

[2] The authors would like to thank R. Mooney's team for having generously supplied the dataset, Gian Marco Laube for his support in implementing the prototype, and the anonymous reviewers for their insightful comments. This work was partially supported by the Swiss National Science Foundation (200021-100149/1)

14. P. Lambrix, M. Habbouche, and M. Prez. Evaluation of ontology development tools for bioinformatics. *Bioinformatics*, 19(12):1564–1571, 2003.
15. V. Lopez, E. Motta, and V. Uren. Poweraqua: Fisching the semantic web. In *3rd European Semantic Web Conference (ESWC 2006)*, pages 393–410, Budva, Montenegro, 2006.
16. V. Lopez, M. Pasin, and E. Motta. Aqualog: An ontology-portable question answering system for the semantic web. In *2nd European Semantic Web Conference (ESWC 2005)*, pages 546–562, Heraklion, Greece, 2005.
17. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Seventh Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR2000)*, pages 483–493, Breckenridge, CO, 2000.
18. D. L. McGuinness and F. van Harmelen. Owl web ontology language overview. W3c recommendation, 2004.
19. R. J. Mooney. Learning semantic parsers: An important but under-studied problem. In *AAAI 2004 Spring Symposium on Language Learning: An Interdisciplinary Perspective*, pages 39–44, Stanford, CA, 2004.
20. H. A. Napier, D. M. Lane, R. R. Batsell, and N. S. Guadango. Impact of a restricted natural language interface on ease of learnng and productivity. *Communications of the ACM*, 32(10):1190–1198, 1989.
21. J. Nielsen. *Usability Engineering*. Academic Press, San Diego/New York, 1993.
22. N. F. Noy, M. Sintek, S. Decker, M. Crubzy, R. W. Fergerson, and M. A. Musen. Creating semantic web contents with protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.
23. A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *8th Intl. Conf. on Intelligent User Interfaces*, pages 149–157, Miami, FL, 2003.
24. J. Preece, Y. Rogers, and H. Sharp. *Interaction Design: Beyond Human-Computer Interaction*. John Wiley and Sons, New York, 2002.
25. E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. Technical report, W3C Candiate Recommendation, 2006.
26. R. Schwitter and M. Tilbrook. Let's talk in description logic via controlled natural language. In *Logic and Engineering of Natural Language Semantics (LENLS2006)*, Tokyo, Japan, 2006.
27. A. Spink, W. Dietmar, B. J. Jansen, and T. Saracevic. Searching the web: The public and their queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–234, 2001.
28. A. Spoerri. Infocrystal: A visual tool for information retrieval management. In *Second Intl. Conf. on Information and Knowledge Management*, pages 11–20, Washington, D.C., 1993. ACM Press.
29. Y. Sure, E. Michael, J. Angele, S. Staab, R. Studer, and D. Wenke. Ontoedit: Collaborative ontology development for the semantic web. In *First Intl. Semantic Web Conf. 2002 (ISWC 2002)*, pages 221–235, Sardinia, Italy, 2002.
30. V. Tablan, T. Polajnar, H. Cunningham, and K. Bontcheva. User-friendly ontology authoring using a controlled language. Research Memorandum CS-05-10, Department of Computer Science, University of Sheffield, 2005.
31. L. R. Tang and R. J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *12th European Conf. on Machine Learning (ECML-2001)*, pages 466–477, Freiburg, Germany, 2001.
32. C. W. Thompson, P. Pazandak, and H. R. Tennant. Talk to your semantic web. *IEEE Internet Computing*, 9(6):75–78, 2005.