A Three-Dimensional Paradigm for Conceptually Scoped Language Technology

Jeroen van Grondelle and Christina Unger

Abstract Language technology is used increasingly for providing speech and text-based interfaces to existing applications and services. However, a number of characteristics of today's language technology make it hard to be adopted by non-linguistically skilled developers. In this paper, we propose a paradigm that conceptually scopes the coverage of the language technology that is adopted into existing applications. It is backed by a three-dimensional approach to modularization of resources that decouples the domains, tasks and languages that need to be supported. We present an implementation of this paradigm based on the ontology-lexicon format *lemon* and Grammatical Framework, and show how the proposed modularity facilitates low impact adoption, through sharing and reuse of technology components and lexical resources on the web.

1 Introduction

Natural language plays an increasingly important role as interface to existing services and data. Social networks, for instance, present updates and news feeds as natural language content, virtual assistants support users by allowing them to query different sources of information and to manipulate them using speech dialogs [Zue and Glass, 2000, Kaljurand and Alumäe, 2012], and business applications allow domain experts to customize the services by creating rules or manage complex configurations using natural language based interfaces [Spreeuwenberg and Healy, 2010, Spreeuwenberg et al., 2012]. The development of language technology that has to support these new applica-

Jeroen van Grondelle
Be Informed, Apeldoorn, The Netherlands, e-mail: j.vangrondelle@beinformed.com

Chistina Unger
CITEC, Bielefeld University, Germany, e-mail: cunger@cit-ec.uni-bielefeld.de

tion scenarios, however, so far has built on objectives and requirements that widely differ from those imposed by their role as interfacing technology, and the consequences still hinder an easy adoption in such scenarios. This can be demonstrated along three main points.

First, an objective of language technology often has been to process unrestricted language. On the one hand, this involves challenges that can be tackled very differently when interfacing with an application. In an unrestricted setting, for example, natural language expressions are highly ambiguous, while in the context of a particular application, they usually have a single, very specific meaning. That is, the application introduces a context that can be exploited for disambiguation. On the other hand, there is a mismatch between the very general, usually surface-oriented meaning representations created in an unrestricted setting, and the demand of aligning language with data and services in the context of a particular application. Again, the interpretation of natural language expressions can be restricted and guided by the underlying application, as it inherently introduces a scope that determines the language fragment that is relevant and meaningful when interfacing with it.

Second, language technology tools and techniques have mainly been developed and used by linguistically trained people. Choosing from the range of available approaches and tools and implementing the selected technology in a specific application requires linguistic expertise typically not found in the companies that develop applications that a language interface is adopted into. Therefore the adoption is costly and requires high upfront investments.

And third, language technology tools often trade precision for additional coverage, while for companies adopting those tools, high precision as well as reliability and predictability become critical, as any misinterpretation can lead to immediate errors in the invocation or execution of the underlying service.

To support the adoption of language technology into existing services and applications by companies with little or no linguistic expertise, we propose a new architecture for language technology that

- is *conceptually scoped* in the sense that it uses the application's conceptualization to scope language technology, and as a consequence limits and tailors all interpreted and generated language to the specific application it is meant to interface with, and
- modularizes the creation and use of resources by clearly separating three dimensions: domains, tasks, and languages.

This shifts the mainstream paradigm of unscoped, monolithic, and therefore costly language technology to a new, strongly modular and inexpensive way of creating and exploiting natural language resources.

2 A Three-Dimensional Paradigm for Conceptually Scoped Language Technology

To address the challenges described above, we propose the following three principles for guiding the development of conceptually scoped, modular language technology.

Scoping natural language through a conceptualization

We propose that the scope of the natural language fragment that is to be supported, for instance through interpretation, generation or translation, should be determined by a conceptualization. Such a conceptualization typically defines the individuals, classes, properties and relations that will be expressed in natural language, independent of the particular representation formalism used, and it should follow from the application or service that language is supposed to interface with. As a result, conceptual scoping grounds any supported natural language utterance in the underlying conceptualization and ensures it to be meaningful within that conceptualization (as, e.g., advocated by Gatius and Rodríguez [1996] and Nirenburg and Raskin [2004]).

This improves on the mainstream paradigm, where conceptualization and attached language technology are often developed independently from each other, and where it is therefore hard to ensure that the conceptual and the linguistic scope of an application are aligned, especially if the conceptualization changes over time.

Automatically generating resources from declarative lexical information

The supported conceptualization has its own lexical aspects, which conventionally have been captured in formalisms that are highly dependent on the type of language technology used. In contrast, by building on a declarative format for specifying lexical information, those lexical aspects can be captured in a technology-neutral way. Technology-specific artifacts, such as grammars, can then be generated by means of a mapping from the declarative lexical representation to the technology-specific formalism.

We therefore propose the pipeline in Figure 1, starting from a conceptualization that is enriched with a declarative specification of the lexical information associated with the given concepts [Reymonet et al., 2007, Montiel-Ponsoda et al., 2008, McCrae et al., 2012, Wróblewska et al., 2012]. The resulting lexical representations are input to an automatic mapping to a language technology-specific resource, such as grammars, phrase tables, or semantic annotations.

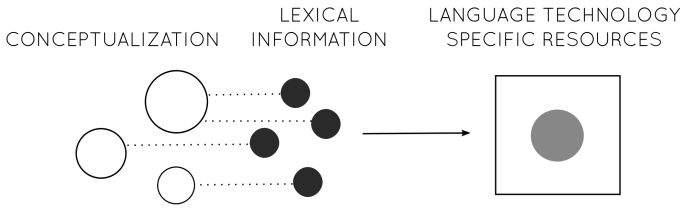

Fig. 1 *Pipeline from conceptualization and lexical information to specific resources*

By automatically generating resources from an intermediate declarative lexical level, the investment into the lexical resources is protected and consistency across the technology-specific resources is guaranteed. Furthermore, the developers of the lexical resources do not need to have expertise in language technology implementations, such as specific grammar formalisms. This lowers the investment required for natural language based applications and furthermore removes the dependency on particular third-party tools. Also, when using declarative lexical formalisms, developers are less likely to make implicit choices concerning particular linguistic theories, which would hamper the reuse of the resources when adopting new technologies that do not agree with these choices.

Decoupling domain and task aspects

Conventionally, a lot of emphasis has been on domain aspects [Martins and de Almeida Falbo, 2008]. But when providing natural language support in an application, the type of tasks supported by that application typically also has linguistic implications in terms of the natural language fragment that needs to be supported. For instance, the task of customer service dialog introduces its own words and sentence structures, independent of the domain. Similarly, task-specific linguistic aspects exist for tasks such as validation of domain ontologies, documentation, etc.

In order to allow for a reuse of task aspects across different domains, we propose the model depicted in Figure 2 to decompose the scope introduced by the underlying applications that language technology interfaces with into three dimensions: the *domain* of the application, specified by some conceptualization, the *task* that the application offers, such as verbalizing domain data for explanation or documentation purposes, or providing online services that include transactions and web forms in terms of the domain, and the *languages* in which the natural language capabilities are offered. The language fragment supported by an application is now defined by a subspace of the resulting cube, involving one or more domains together with one or more tasks and spanning one or more languages.

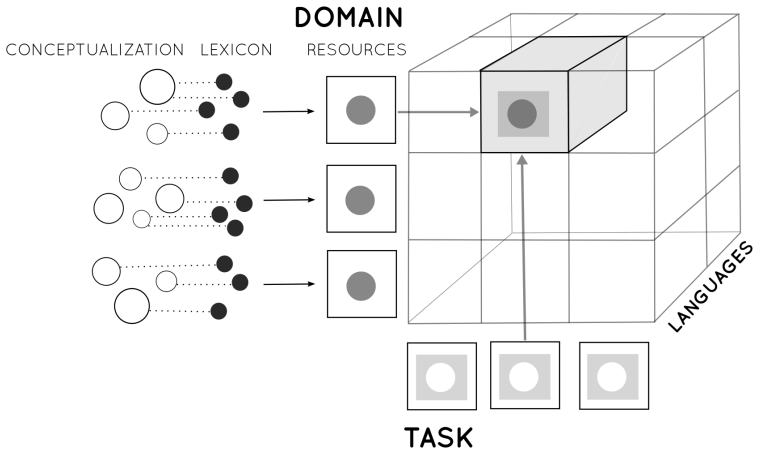

Fig. 2 *Three dimensional model for conceptually-scoped language technology*

This orthogonal modularization of domain and task aspects supports specification of the conceptualization and lexical information per dimension, i.e. specifying domains independent from tasks and vice versa. The dimensions can then be freely combined by choosing the particular domains, tasks and languages supported for a specific application. This allows not only for the reuse of already existing conceptualizations, such as adding new tasks to an existing domain or reusing task conceptualizations across different domains, but steadily increases the return on investment, since the more of these building blocks already exist, the easier and faster it is to plug them together to build new applications.

The importance of separating domain and task conceptualizations has also been noted, e.g., by Guarino [1997] and Mizoguchi et al. [1995].

3 Proof of Concept in the Context of the Multilingual Semantic Web

As a proof of concept of the three-dimensional paradigm proposed, we implement a dialog-oriented natural language interface based on these principles, using a stack of technologies suitable in the context of the multilingual Semantic Web, and show that it supports typical scenarios in the incremental adoption of language technology.

3.1 Implementation

For capturing conceptualizations and lexicalizations, we build on existing Semantic Web standards, in particular OWL and RDF. Since these standards by their very nature enable linking and sharing of data, this supports the reuse of modules and facilitates an ecosystem of language technology resources, as discussed in Section 4 below.

The domain conceptualization is captured as an ontology in the standard ontology format OWL [McGuinness and van Harmelen, 2004]. In order to be able to associate linguistic information with concepts in an ontology, the lexical component is implemented using *lemon* [McCrae et al., 2012], a model for the declarative specification of multilingual ontology lexica in RDF. It allows lexical data to be published, shared and interlinked on the web, and thus fits very well with our approach's strong emphasis on modularity. Furthermore, it is independent of a particular linguistic theory or grammar formalism. In the following, we use Grammatical Framework [Ranta, 2011] as target grammar formalism, benefiting from its inherent modularity and its support for more than 30 languages, which allows for very fast and effortless porting across those supported languages.

The instantiation of the pipeline from conceptualization and lexicalization to a specific grammar thus starts from an OWL ontology, then requires the creation (or reuse) of an ontology-lexicon for the target languages, specifying lexicalization of the ontology elements in those particular languages, and then relies on the automatic generation of multilingual grammars from that lexicon.

In order to percolate modularity up to the grammar level, we implement application grammars as being composed of three modules, as depicted in Figure 3: a domain- and task-independent *core* grammar, an automatically generated *domain* grammar, and an accompanying *task* grammar.

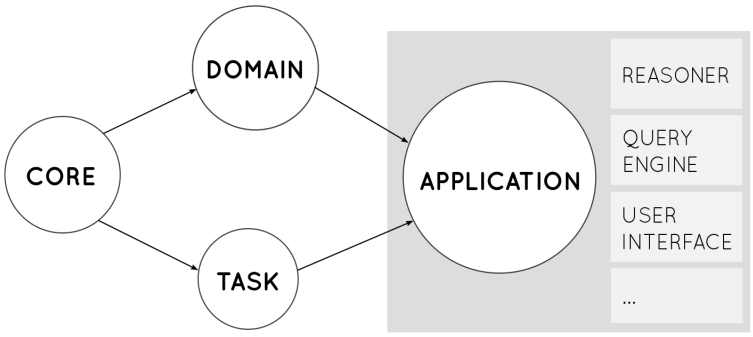

Fig. 3 *Grammar modularity. Arrows indicate grammar inheritance.*

The core grammar comprises domain- and task-independent expressions, especially closed class expressions such as determiners, pronouns, auxiliary verbs, coordination expressions and negation. It is created manually and can be reused for every domain and task. It provides an independent basis on which both domain and task grammars build, therefore acting as a decoupler between them.

The domain grammar extends the core with expressions that are automatically generated from a given ontology-lexicon, following the pipeline in Figure 1. The task grammar, on the other hand, extends the core with task-relevant expressions. As of now it is created manually, but carrying over the grammar generation pipeline from the domain to the task dimension and thereby also allowing for the automatic generation of task grammars constitutes future work, see Section 5 below. The fact that domain and task grammars are constructed independently from each other, together with the fact that they share the core grammar as their basis, allows for a free and smooth combination of any domain with any task.

The application grammar finally combines core, domain and task grammars, and furthermore allows for application-specific extensions or fine tuning. The final application grammar is used for the specific purpose of the application, which possibly interfaces it with additional modules such as a reasoner, a query engine, or a user interface.

3.2 Typical Adoption Scenarios

In the following we illustrate typical adoption scenarios, in particular decoupling domain and task aspects, incorporating new domains and tasks, and adding further languages. The mentioned resources can be accessed at `http://purl.org/3dlt/home`.

3.2.1 Decoupling Domain and Task Aspects

We start by building an application grammar for customer service dialog in the flight travel domain in English. That is, given a conceptualization of flight travel, we want to construct a grammar that captures utterances such as the following ones:

- Show me all flights from Boston to Detroit.
- Which airlines fly to San Francisco?
- I want to travel to New York tomorrow.
- When do you want to depart?
- Do you need a hotel in New York?

The domain conceptualization is modelled as an OWL ontology that was built in the context of the PortDial project[1], based on terms used in a corresponding *Airline Travel Information System* (ATIS) domain grammar [PortDial Consortium, 2013]. It is organized around the concept of a trip, which consists of flights, hotel stays and car rentals. Flights in turn are composed of flight legs and are connected to their arrival and departure as well as the operating airline. As an example, Figure 4 shows a small part of the ontology, capturing flights, the city of their departure and their operating airline.

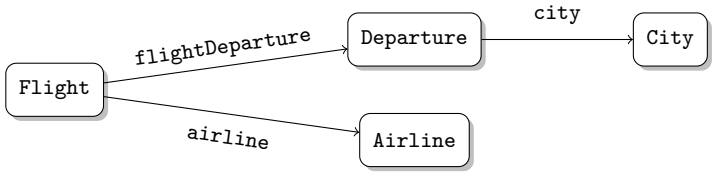

Fig. 4 *Conceptualization of flights with their departure city and operating airline.*

Connected to the ontology is an ontology-lexicon that specifies how the classes, properties and individuals are verbalized in a specific language. The classes `Flight` and `City`, for example, are expressed in English using the nouns flight and city, while `Departure` is an auxiliary construct that a user would probably not address directly. The latter also holds for both the properties `flightDeparture` and `city`: On their own they are not relevant to the user, but what is relevant is their composition, connecting flights to the city of their departure. The property chain `flightDeparture` ∘ `city` can be verbalized as the verb chunk to depart from, the verb and to leave, or as the noun chunk flight from. A natural verbalization of the property `airline` is by means of the verb to operate. Examples for lexical patterns specifying those verbalizations are listed in Figure 5, using a catalogue of *lemon* design patterns [McCrae and Unger, this volume] that was created in order to relieve lexicon engineers from the need to understand and write RDF as well as to support them in the construction of lexical entries. All patterns specify a canonical form (possibly together with additional inflectional forms) as well as a reference to the particular ontology concept they denote. The verb patterns moreover give a mapping from semantic to syntactic arguments: In the case of to operate the subject of the denoted property (a flight) corresponds to the direct object in the syntactic structure, and the object of the denoted property (an airline) corresponds to the syntactic subject, like in Pan Am operates flight 27B-6, while in the case of to depart the subject of the denoted property chain (a flight) corresponds to the syntactic subject, and the object of the denoted property chain (a city) corresponds to a prepositional object in the syntactic structure, marked with the preposition from.

[1] `https://sites.google.com/site/portdial2/`

```
ClassNoun("flight",:Flight) with plural "flights"
ClassNoun("city",  :City)   with plural "cities"

StateVerb("operate",:airline,
      propSubj=DirectObject,
      propObj=Subject)

StateVerb("depart",:flightDeparture∘:city,
      propSubj=Subject,
      propObj=PrepositionalObject("from"))
```

Fig. 5 *Lexical patterns for the nouns* flight *and* city *as well as the verbs* to depart from *and* to operate.

In a similar way, the lexicon specifies alternative verbalizations of the same concepts, such as to leave from or flight from, as well as all relevant verbalizations of other ontology concepts. In exactly the same way, lexicalizations of instances can be given, e.g. verbalizing the individual `Boston` by its name Boston, and the individual `John_F_Kennedy_International_Airport` as JFK.

Once an ontology-lexicon is constructed, it is used for the automatic generation of a domain grammar. For this we build on *lemongrass*[2], a Python script for mapping *lemon* lexica to different grammar formats, including Grammatical Framework (GF). GF distinguishes abstract and concrete syntax. The abstract syntax is a type-theoretical framework for specifying abstract concepts in a language-independent manner. These concepts are usually semantic ones, which makes it possible to, e.g., use the abstract syntax to represent ontologies Angelov and Enache [2012]. A concrete syntax is a mapping from abstract syntax concepts to linearizations of those concepts in a particular language. Based on the concepts in the ontology, *lemongrass* constructs an abstract syntax, and from the morphosyntactic information specified in the lexicon, *lemongrass* instantiates templates for constructing a corresponding concrete syntax. The result is a domain grammar that, together with the domain-independent expressions from the core grammar, captures phrases like all flights to Boston, and the flight is operated by an airline which serves JFK.

Since the domain conceptualization does not cover any task-relevant concepts, neither the lexicon nor the resulting grammar comprises expressions specific for customer service dialogs. Providing such expressions is the job of the task grammar, for example specifying constructions for requesting and offering information, as well as dialog constructions such as greetings and expressions for agreement or disagreement, possibly taking into account parameters like formal vs. informal speech.

The final application grammar is then composed of the core grammar, an automatically generated domain grammar for the flight travel domain, and a (for now manually constructed) task grammar for user service dialogs.

[2] `https://bitbucket.org/chru/lemongrass`

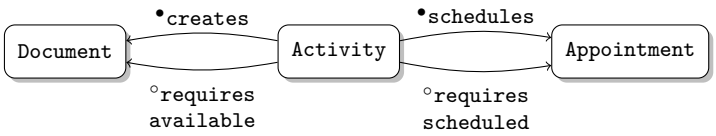

Fig. 6 *Conceptualization of activities and their preconditions and postconditions, where* ° *marks precondition relations and* • *marks postcondition relations.*

Combining these three grammar modules, the covered language fragment includes utterances like give me all flights to Boston, and which airlines operate flights from Boston to Denver.

3.2.2 Porting to a New Domain

Porting the above dialog application to another domain requires a conceptualization of that new domain, together with lexical information from which a new domain grammar can be generated. Depending on the size and complexity, lexicon creation can be very labour-intensive and thus would greatly benefit from semi-automatic methods [Walter et al., 2013] and an ecosystem of resources as described in Section 4 below. Because of the independence of grammar modules, core and task grammars remain unchanged.

We illustrate the domain porting by an example from the business processes domain, centered around activities and their preconditions and postconditions [van Grondelle and Gülpers, 2011]. Figure 6 shows an instantiation for the particular case of housing benefit requests, where relevant activities are, for instance, assessing a request, planning a meeting, or publishing a decision. Preconditions of such activities comprise the availability of some document or a scheduled appointment, while postconditions include the creation of a document, e.g. a confirmation or rejection letter.

In the ontology, both preconditions and postconditions are modelled as object properties, with `creates` and `schedules` as subproperties of the postcondition property, and `requires` as subproperty of the precondition property. States like `available` and `scheduled` are modelled as classes. The composed precondition relations `requires available` and `requires scheduled` are then defined as properties with a range comprising individuals from the union of, e.g., `Document` and `Available`. An example of such a definition is given in Figure 7.

Similar to the flight travel domain, a corresponding ontology-lexicon specifies how the classes, relations and instances are verbalized. The precondition `requires`, for example, can straightforwardly be expressed using the verb to require, as in the following example:

- The assessment of the request requires that the customer visit is scheduled.

An example of a lexicon pattern for this verbalization as well as one for the class `Available` is given in Figure 8.

Coupling the housing benefit domain grammar with the customer service dialog task used above then allows for the generation of questions and requests such as Which documents are required for assessing the request? and We need the confirmation letter.

3.2.3 Incorporating New Tasks

Analogously to replacing one domain by another, we can also replace one task by another. For instance, for the purpose of creating explanatory texts, a task grammar could contain constructions for combining fact verbalizations using because, therefore, but and other conjunctions, as well as expressions for putting emphasis on particular aspects. Combining such a documentation task grammar with the housing benefit domain grammar can cover expressions such as the following ones:

- Especially the customer visit is required.
- A confirmation letter was created. Therefore the activity of assessing the request is completed.

The new task can of course also be combined with the flight travel domain, covering expressions such as the following ones:

- Especially JFK is served by most airlines.
- A flight from Los Angeles to San Francisco takes one hour. Therefore there is no meal.

```
:Available          a owl:Class .

:precondition       a owl:ObjectProperty .

:requires           a owl:ObjectProperty ;
    rdfs:subPropertyOf :precondition .

:requires_available a owl:ObjectProperty ;
    rdfs:subPropertyOf :requires ;
    rdfs:domain :Activity ;
    rdfs:range [ owl:intersectionOf (:Document :Available) ].
```

Fig. 7 *Definition of the precondition relation* `requires available`*, based on the general precondition property* `requires` *and the class* `Available`*.*

```
StateVerb("require",:requires)

IntersectiveAdjective("available",   :Available)
IntersectiveAdjective("unavailable",:Available⁻¹)
```

Fig. 8 *Lexical patterns for the verb* requires *as well as the intersective adjectives* available *and* unavailable*, where* `Available`$^{-1}$ *denotes the complement class of* `Available`*.*

3.2.4 Adding Further Languages

Extending an application to other languages requires porting both the lexicalizations and the lexicon-to-grammar mapping.

First, the domain lexicon needs to be ported to the target language. This process can exploit automatic methods for ontology lexicalization [Walter et al., 2013], label translation methods [Mejía et al., 2009, McCrae et al., 2011] and linguistic resources such as BabelNet [Navigli and Ponzetto, 2012]. Figure 9 shows Dutch versions of the flight travel lexicalizations given in Figure 5 above. Since Dutch is very close to English, the lexicalizations only differ in their form and in the specification of gender in the case of nouns.

```
ClassNoun("vlucht",:Flight) commonGender
                            with plural "vluchten"
ClassNoun("stad",   :City)  commonGender
                            with plural "steden"

StateVerb("opereren",:airline,
      propSubj = DirectObject,
      propObj  = Subject)

StateVerb("vertrekken",:flightDeparture∘:city,
      propSubj = Subject,
      propObj  = PrepositionalObject("van"))
```

Fig. 9 *Dutch lexical patterns for flight travel concepts.*

Second, the lexicon-to-grammar mapping and the core grammar module needs to be ported to the target language. The involved effort strongly depends on the grammar formalism and the multilingual resources available in that formalism. In the case of our implementation using GF, porting a grammar to another language is almost trivial for all languages for which GF provides resource grammars, i.e. implementations of low-level morphosyntactic operations. This is the case for about 30 languages from a variety of language families. Mapping the core grammar module to Dutch and German required about 10 lines of GF code each, and extending *lemongrass* with templates for additional concrete syntaxes for those languages required a similarly low amount of effort.

The grammar constructed from the Dutch flight travel lexicon, together with the Dutch core and task grammar modules, then covers utterances such as the following ones:

- Toon alle vluchten vanaf Detroit naar Boston.
- Welke luchtvaartmaatschappijen vliegen naar San Francisco?
- Ik wil morgen naar New York reizen.

4 An Ecosystem for Language Technology

The architecture presented is extremely modular, both in terms of technologies and resources. This provides new ways of sharing and marketing language technology, as granular components can be developed independently and can then be shared, reused and composed into language technology-based solutions, thereby facilitating an ecosystem of cooperating language technology producers and consumers.

In addition to language resources like declarative lexical resources for domains and tasks, a number of different kinds of components could be shared:

- generic domain and task conceptualizations
- technologies to mine and extend lexical resources
- technology mappings from declarative lexical resources to technology-specific formalisms, such as different grammar formalisms, phrase tables, semantic annotations [Davis et al., 2011], etc.

Being able to reuse technology and lexical resources at a granular level provides non-linguistically trained developers with a low impact adoption path of language technologies into existing applications and solutions. Initial support for natural language can be added at low cost, as default lexical and grammar resources are available for reuse, as are tools to create and enrich those resources. Optimization and customization can then be performed as expertise grows.

The open standards of the Semantic Web provide a very suitable way to implement such an ecosystem, as it supports the publishing and sharing of resources and services on the web, based on Semantic Web formalisms and tools. Examples are the Linguistic Linked Data [Chiarcos et al., 2012] cloud[3], which forms a growing ecosystem of interlinked language resources such as dictionaries and lexica, and the Language Grid [Murakami et al., this volume], which offers an architecture for sharing and composing language services.

A different way to exploit the modularity of the resources is creating extensible, novel end user services, as shown in Figure 10. Imagine a virtual assistant, presumably on a smart phone, that could easily be extended by app developers with new capabilities and that allows consumers to create

[3] `http://linguistics.okfn.org/resources/llod/`

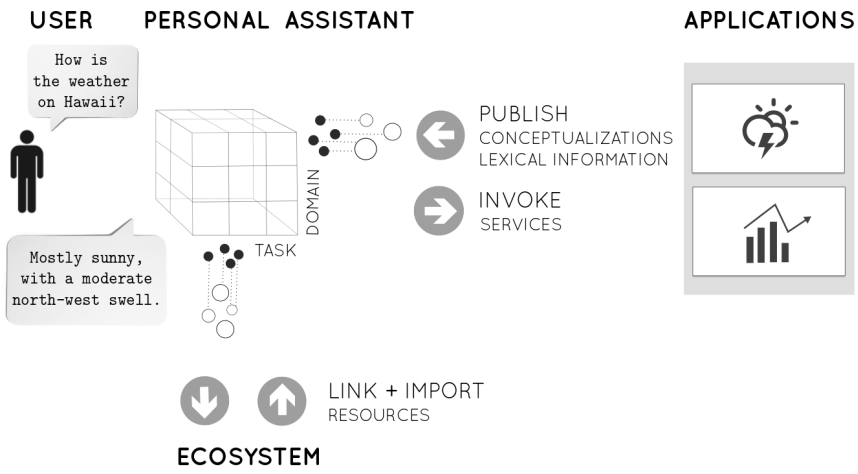

Fig. 10 *A virtual assistant as natural language interface to applications.*

their own personal virtual assistant supporting services of interest to them, and as a consequence, covering exactly the range of dialog needed for those selected services. The architecture we presented in this paper could be the basis of a software development kit that allows app developers to associate their apps with domain and task conceptualizations and lexicalizations to allow for instance the phone's standard virtual assistant to disclose the app's services using voice dialog. For instance, a weather app could come with a conceptualization and lexicalization of the weather domain, allowing the consumer to query the phone's virtual assistant for the weather situation, possibly using a standard vocabulary for querying.

5 Conclusion and Future Work

In order to support the adoption of language technology into existing services and applications, especially by companies with little or no linguistic expertise, we proposed a new paradigm for the creation and use of language technology resources. Starting from a conceptualization that scopes the supported language fragment to exactly those expressions and constructions relevant for the application in question, we exploited declarative lexical information for specifying verbalizations of concepts. On both levels, conceptual and lexical, we clearly separated domain and task aspects. Further, lexical representations served as input for the automatic generation of language technology resources, thereby removing both the need for expertise in specific formats and the dependence on particular implementations of them.

As proof of concept, we provided an implementation based on Semantic Web standards, creating GF grammars from *lemon* lexicalizations attached to an underlying OWL conceptualization of a domain, showing that it supports typical adoption scenarios.

A limitation to be addressed in future work is that in the given implementation, task grammars were still constructed manually. This mirrors the fact that the conceptualizations and lexica already present on the web so far mainly focus on domains, whereas the task that is supported is often implicitly assumed to be querying. Conceptualization of other tasks as well as multilingual lexical information for verbalizing them are still widely lacking. We therefore aim at a general conceptualization of different tasks and, if necessary, an extension of the *lemon* model for task verbalizations.

Furthermore, we plan to explore how the proposed paradigm can be applied to other areas of language technology, for example generating phrase tables for machine translation, possibly building on the same Semantic Web standards for conceptualizations and lexicalizations.

This will lift the proposed three-dimensional architecture to its full potential, enabling the reuse of multilingual lexical resources for domains and tasks across the web and allowing the application of these resources in a wide range of language technologies, moving towards an ecosystem for language technology.

Acknowledgments

This work was partially funded in the EU projects PortDial (FP7-296170), Monnet (FP7-248458) and MOLTO (FP7-247914). We also want to thank the organizers of the Dagstuhl seminar, where many of the ideas in this paper took form, especially Philipp Cimiano for numerous invaluable discussions. We are also indebted to Aarne Ranta, Jouri Fledderman and Frank Smit.

References

Victor W. Zue and James R. Glass. Conversational interfaces: advances and challenges. *IEEE Special Issue on Spoken Language Processing*, 88(8): 1166–1180, 2000.

Kaarel Kaljurand and Tanel Alumäe. Controlled natural language in speech recognition based user interfaces. In *Controlled Natural Language*, volume 7427 of *LNCS*, pages 79–94. Springer, 2012.

Silvie Spreeuwenberg and Keri Anderson Healy. SBVR's approach to controlled natural language. In *Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, pages 155–169, 2010.

Silvie Spreeuwenberg, Jeroen van Grondelle, Ronald Heller, and Gartjan Grijzen. Using CNL techniques and pattern sentences to involve domain experts in modeling. In *Preoceedings of the Workshop on Controlled Natural Language (CNL 2010)*, pages 175–193, 2012.

M. Gatius and H. Rodríguez. A domain-restricted task-guided natural language interface generator. In *Proceedings of the Second Edition of the Workshop Flexible Query Answering Systems (FQAS'96)*, 1996.

Sergei Nirenburg and Viktor Raskin. *Ontological Semantics*. MIT Press, 2004.

A. Reymonet, J. Thomas, and N. Aussenac-Gilles. Modelling ontological and terminological resources in OWL DL. In *Proceedings of the OntoLex07 Workshop at the 6th International Semantic Web Conference (ISWC 2007)*, 2007.

E. Montiel-Ponsoda, G. Aguado de Cea, A. Gómez-Pérez, and W. Peters. Modelling multilinguality in ontologies. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*, pages 67–70, 2008.

John McCrae, Guadelupe Aguado de Cea, Paul Buitelaar, Philipp Cimiano, Thierry Declerck, Asunción Gómez-Pérez, Jorge Garcia, Laura Hollink, Elena Montiel-Ponsoda, and Dennis Spohr. Interchanging lexical resources on the Semantic Web. *Language Resources and Evaluation*, 46(4):701–719, 2012.

Anna Wróblewska, Grzegorz Protaziuk, Robert Bembenik, and Teresa Podsiadły-Marczykowska. Lexo: A lexical layer for ontologies – design and building scenarios. *Studia Informatica*, 33(1), 2012.

Aline Freitas Martins and Ricardo de Almeida Falbo. Models for representing task ontologies. In Fred Freitas, Heiner Stuckenschmidt, Sofia Pinto, Andreia Malucelli, and Oscar Corcho, editors, *Proceedings of the 3rd Workshop on Ontologies and their Applications (WONTO 2008)*, 2008.

Nicola Guarino. Understanding, building and using ontologies. *International Journal of Human-Computer Studies*, 46(2–3):293–310, 1997.

Riichiro Mizoguchi, Yuri Tijerino, and Mitsuru Ikeda. Task analysis interview based on task ontology. *Expert Systems with Applications*, 9(1):15–25, 1995.

Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language overview. *W3C Recommendation*, 2004.

Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, 2011.

PortDial Consortium. D2.1 Free Data Deliverable, 2013. `https://sites.google.com/site/portdial2/deliverables-publications/free-data-deliverable`.

John McCrae and Christina Unger. Design patterns for engineering the ontology-lexicon interface. this volume.

Krasimir Angelov and Ramona Enache. Typeful ontologies with direct multilingual verbalization. In Michael Rosner and Norbert E. Fuchs, editors,

Controlled Natural Language, volume 7175 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2012.

Sebastian Walter, Christina Unger, and Philipp Cimiano. A corpus-based approach for the induction of ontology lexica. In *Proceedings of the 18th International Conference on Application of Natural Language to Information Systems (NLDB 2013)*, 2013.

Jeroen van Grondelle and Menno Gülpers. Specifying flexible business processes using pre and post conditions. In *Practice of Enterprise Modeling*, volume 92 of *Lecture Notes in Business Information Processing*, pages 38–51, 2011.

Mauricio Espinoza Mejía, Elena Montiel-Ponsoda, and Asunción Gómez-Pérez. Ontology localization. In *Proceedings of the Fifth International Conference on Knowledge Capture (K-CAP 2009)*, pages 33–40, 2009.

John McCrae, Mauricio Espinoza, Elena Montiel-Ponsoda, Guadalupe Aguado-de Cea, and Philipp Cimiano. Combining statistical and semantic approaches to the translation of ontologies and taxonomies. In *Proceedings of the Fifth Workshop on Syntax, Structure and Semantics in Statistical Translation (SSST-5)*, pages 116–125, 2011.

Roberto Navigli and Simone Paolo Ponzetto. BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012.

Brian Davis, Fadi Badra, Paul Buitelaar, Tobias Wunner, and Siegfried Handschuh. Squeezing lemon with GATE. In *Proceedings of the 2nd International Workshop on the Multilingual Semantic Web (MSW 2011), Workshop at the 10th International Semantic Web Conference (ISWC 2011)*, 2011.

Christian Chiarcos, Sebastian Nordhoff, and Sebastian Hellmann, editors. *Linked Data in Linguistics: Representing and Connecting Language Data and Language Metadata*. Springer, 2012.

Yohei Murakami, Donghui Lin, and Toru Ishida. Service oriented architecture for interoperability of multi-language services. this volume.