# GATE in NeOn: mutual support for ontology lifecycle development and natural language processing

Diana Maynard and Wim Peters and Adam Funk

Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello
S1 4DP, Sheffield, UK
`{diana,wim,a.funk}@dcs.shef.ac.uk`

**Abstract.** This paper describes the implementation of an approach to modelling the dynamics of the propagation of textually derived semantic information, in particular with respect to networked ontologies. On the one hand, new ontologies may be generated automatically from textual data, or existing ontologies may be modified or extended according to new evidence. This can cause problems for large or networked ontologies, where only a small section of the ontology may be modified, and where multiple users may be working with the ontologies simultaneously. Furthermore, interoperability issues may occur where different people are using different tools: for example, some users may be automatically annotating texts and generating new ontologies using NLP techniques such as those found in GATE, while other users may be applying different techniques such as using the various plugins of the NeOn toolkit. In this paper, we describe a plugin for the NeOn toolkit which uses both automatic and manual methods for generating and modifying ontologies on the fly, and which incorporates change logging to ensure compatibility between ontology versions. We thus combine support for both ontology lifecyle development and NLP engineers in a single mechanism that brings together an established language engineering community (GATE) with a newly emerging ontology community (NeOn).

**Key words:** natural language processing, ontology lifecycle, ontology generation, ontology evolution

## 1 Introduction

Ontology evolution is increasingly acquiring research momentum in the Semantic Web field. This is due to the fact that ontologies, forming the backbone of Semantic Web systems, need to be kept up-to-date so that ontology-based systems remain usable. In this work, we describe an approach to modelling the dynamics of ontology and metadata change in text, motivated by the need to propagate changes in an ontology to its instances and properties, and vice versa.

For example, if a user deletes concepts from the ontology, it is important to have a mechanism for dealing with any associated semantic metadata, in order not to lose vital information. If a user adds new concepts to the ontology, then it may be necessary to return to the text to check whether additional instances can be found which should be used to populate these new concepts in the ontology. We call this the top-down approach to ontology change. Furthermore, not only are ontologies dynamic and subject to structural change, but so are the texts and instances from which the ontologies may be derived. If we get additional relevant textual material and/or find new instances in that text, it may be necessary to modify the ontology to take into consideration this new information (for example, adding new concepts or new relations between existing concepts in the ontology). We call this the bottom-up approach to ontology change.

Our top-down approach enables changes made to the ontology to be propagated to the metadata so that as little information as possible is lost. For example, when a concept is deleted from the ontology, usually any semantic metadata (instances) belonging to that concept would be deleted with it, but this is not always desirable behaviour because often we would prefer to reclassify the instance at a more general level. Our approach is based on a methodology and implementation for change propagation, that makes use of an ontology change typology [1].

Our bottom-up approach enables an existing ontology to be augmented as a result of new textual data. We have developed a generic GATE application[1] called SPRAT (Semantic Pattern Recognition and Annotation Tool) which uses ontology-based information extraction techniques [2, 3] to generate ontological information from unstructured text and either create a new ontology from scratch or augment an existing ontology with new entities. The ontologies generated contain classes, subclasses and properties. These are generated by identifying entities and relations in the text, using pattern-based techniques, augmented with deeper semantic information and selectional restrictions. These techniques are described more fully in Section 2.1. We have tested this application on Wikipedia texts about animals, as these are highly productive in terms of ontological information generated and do not require a domain expert to verify the results. This generic application can also be tuned to specific domains by making use of pre-existing domain ontologies. For example, we have developed another application called SARDINE, targeted at the fisheries domain, which makes use of a large fisheries ontology developed by FAO[2]. This extracts information from fisheries texts and augments their ontology and databases with new facts. For example, we identify relations describing habitats of particular fish species (e.g. Atlantic cod are fished in the Gulf of Maine), and alternative names for fish that may not be in their ontology. Section 2.1 provides some further explanation as to why ontologies may need augmenting.

These applications represent a typical situation where NLP (natural language processing) techniques can assist in the development of Semantic Web technol-

[1] General Architecture for Text Engineering http://gate.ac.uk
[2] Food and Agriculture Organization of the United Nations http://www.fao.org

ogy. The value of NLP systems is to bring some kind of structured order to the chaotic representation of facts that is natural language. Typical web pages and other text-based documents are designed to be easily read and understood by humans, but are difficult for machines to process. NLP techniques provide the link between unstructured text and an ontological representation of facts which can then be used for automatic processing for any number of real world applications, such as business intelligence, information finding, question answering and so on. There are many NLP tools used in both research and in the world of business, of which GATE is one of the most widely used and well known. There are also a variety of tools and systems providing ontology liffecycle support, of which the NeOn toolkit is one newly emerging example. However, one of the current research challenges is to provide better interoperability between these two rather different kinds of systems. For example, while GATE provides ontology support, this is very limited compared with ontology editors and frameworks such as Protégé[4] and the NeOn toolkit.

Both the top-down and bottom-up approaches combine support for ontology lifecyle development and NLP engineers. The ontology refinement and change log mechanisms enable ontological changes to be made in GATE by language engineers and domain experts who are interested in acquiring new information to populate an ontology, without worrying about what this information may later be used for or what other tools may be required to run on the data. These people will use GATE to analyse the documents and/or create the data, but are not necessarily interested in ontology management. On the other hand, ontologists who are not concerned with where the data comes from, but only what use they will make of it, do not have to worry about how to get hold of the relevant information in the first place, and do not need to worry about language engineering tools.

## 2 Ontology acquisition using SPRAT

In this section, we describe the bottom-up approach to ontology change, using the SPRAT application. We combine aspects from traditional named entity recognition, ontology-based information extraction and relation extraction, in order to identify patterns for the extraction of a variety of entity types and relations between them, and to re-engineer them into ontological concepts, instances and properties. The application is developed in GATE. Basic linguistic patterns are first identified, and then enhanced with deeper semantic knowledge. The NEBOnE plugin for GATE is used in order to generate the new ontological entities on the fly: this is described in more detail in Section 2.3.

SPRAT makes use of a number of pre-existing GATE resources for preprocessing the text with linguistic information, such as tokenisation, part-of-speech tagging, simple noun phrase and verb phrase chunking, and named entity recognition. It also includes lexical resources containing semantic classes from WordNet [5] and VerbNet [6], which enable the incorporation of deeper semantic information. This allows us to (i) look for verbal patterns connecting terms in a sentence, using the ANNIC plugin in GATE [7] and (ii) to restrict the kinds of relation extracted. For example, we can restrict the kinds of entities that have body parts associated with them to animals and humans.

## 2.1 Identifying Linguistic Patterns

Identifying ontological concepts and/or relations in text requires a slightly different strategy from that used in traditional named entity recognition and ontology-based information extraction methods [2]. While we can still make use of known terms (either via a gazetteer or by accessing the class, instance and property labels in an existing ontology), this is often not sufficient for a variety of reasons:

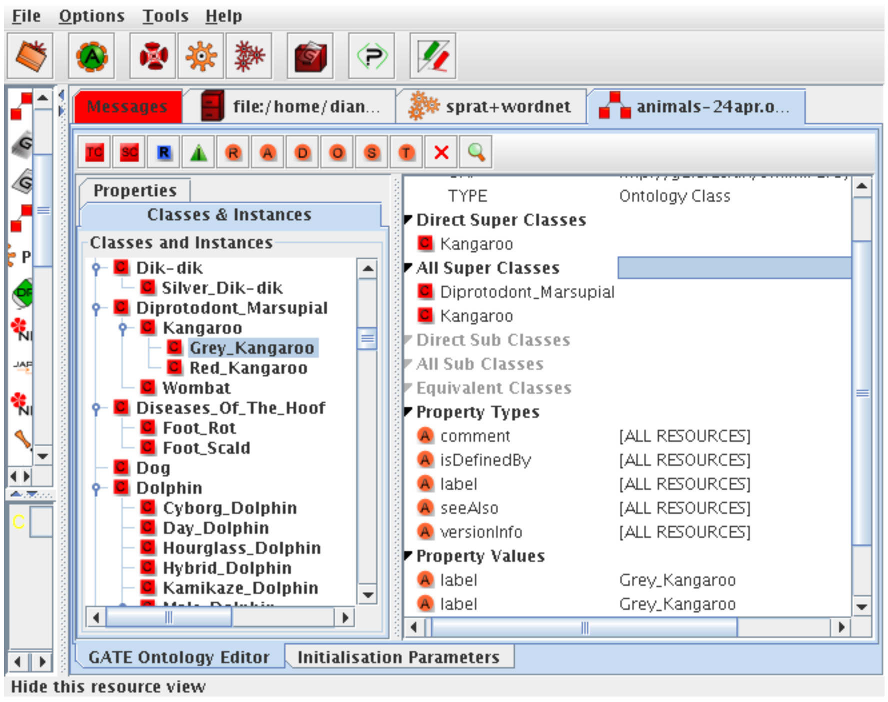
- the concept may not be in the ontology already;
- the concept may exist in the ontology only as a synonym or linguistic variation (singular instead of plural, for example);
- the concept may be ambiguous;
- only a superclass of the concept may exist in the ontology.

We therefore need to make more use of linguistic patterns and also contextual clues, rather than relying on gazetteer lists. We have identified three sets of patterns which can help us identify concepts, instances and properties to extend the ontology: the well-known Hearst patterns [8]), the Lexico-Syntactic Patterns developed in the NeOn project corresponding to Ontology Design Patterns [9], and some new contextual patterns defined by us [10]. We have extended the patterns described in [10] to incorporate further semantic restrictions and to include more properties. An example of a linguistic pattern is the following:
**Add a new class.**
Here we postulate that an unknown entity amidst a list of known entities is likely to be also an entity of the same type. For example, if we have a list of classes of fish, and there is an unknown noun phrase in amongst the list, we can presume that this is also a class of fish. To decide where to add this new class in the ontology, we can look for the Most Specific Common Abstraction (MSCA) of all the other items in the list (i.e. the lowest common superclass of all the classes in the list) and add the new entity as a subclass of this class.
**Example:** *Hornsharks*, *leopard sharks* and *catsharks* can survive in aquarium conditions for up to a year or more.
where *hornshark* and *leopard shark* are classes in the ontology and *catshark* is unknown, so we can recognise *catshark* as a subclass with the same parent as that of *hornshark* and *leopard shark*, in this case *shark*.

An example of selectional restrictions placed on a relation is that a property must be restricted to the following semantic type from WordNet: plant, shape, food, substance, object, body, animal, possession, phenomenon, artifact. Experiments with this restriction improved precision from around 35% to 75%, although unsurprisingly, recall dropped a little.

### 2.2 Implementation of patterns

The patterns are implemented in GATE as JAPE rules [11]. On the left hand side of the rule is the pattern to be annotated. This consists of a number of pre-existing annotations which have been created as a result of pre-processing components (such as POS tagging, gazetteer lookup and so on) and earlier JAPE rules. The right hand side of the rule invokes NEBOnE and creates the new items in the ontology, as well as adding annotations to the document itself. This part of the rule first gets the relevant information from the annotations (using the labels assigned on the LHS of the rule), then adds the new information to the ontology and finally adds annotations to the entities in the document. NEBOnE is responsible also for ensuring that the resulting changes to the ontology are wellformed: this is described in more detail in Section 2.3. Figure 1 shows a screenshot from GATE of an ontology created from a selection of wikipedia texts about animals.



```
Dik-dik
    Silver_Dik-dik
Diprotodont_Marsupial
    Kangaroo
        Grey_Kangaroo
        Red_Kangaroo
    Wombat
Diseases_Of_The_Hoof
    Foot_Rot
    Foot_Scald
Dog
Dolphin
    Cyborg_Dolphin
    Day_Dolphin
    Hourglass_Dolphin
    Hybrid_Dolphin
    Kamikaze_Dolphin
```

| | |
|---|---|
| TYPE | Ontology Class |
| **Direct Super Classes** | |
| Kangaroo | |
| **All Super Classes** | |
| Diprotodont_Marsupial | |
| Kangaroo | |
| Direct Sub Classes | |
| All Sub Classes | |
| Equivalent Classes | |
| **Property Types** | |
| comment | [ALL RESOURCES] |
| isDefinedBy | [ALL RESOURCES] |
| label | [ALL RESOURCES] |
| seeAlso | [ALL RESOURCES] |
| versionInfo | [ALL RESOURCES] |
| **Property Values** | |
| label | Grey_Kangaroo |
| label | Grey_Kangaroo |

**Fig. 1.** Generated ontology in GATE

## 2.3 NEBOnE

The SPRAT application uses the specially developed NEBOnE plugin for GATE in order to generate the changes to the ontology. NEBOnE (Named Entity Based ONtology Editing) is an implementation for processing natural language text and manipulating an ontology. It is derived from the CLOnE plugin [12] for GATE.

In CLOnE, input sentences are analysed deterministically and compositionally with respect to a given ontology, which the software consults in order to interpret the input semantics. CLOnE allows users to design, create, and manage information without knowledge of complicated standards (such as XML, RDF and OWL) or ontology engineering tools. It is implemented as a simplified natural language processor that allows the specification of logical data for semantic knowledge technology purposes in normal language, but with high accuracy and reliability. The components are based on GATE's existing tools for IE (information extraction) and NLP. Because the parsing process is deterministic, accuracy is not an issue: as long as the user specifies their input in correct controlled language, the system always produces correct output.

NEBOnE is based on the same underlying principles as CLOnE and is realised as another GATE plugin. The idea behind NEBOnE is that once a text has been annotated using Named Entity recognition techniques, these annotations can be used to generate new concepts, instances and properties in the ontology. CLOnE uses so-called *chunks* from the input sentences as candidates for inclusion in the ontology as classes, instances and properties: these are noun phrases previously created by a chunker in GATE. In NEBOnE, however, a *chunk* can be any annotation previously created, and does not need to correspond to a noun phrase, thereby ensuring a great deal more flexibility. For example, some classes may have a shorter span than a noun phrase, e.g. "shark" occurring in the context of "leopard shark" (where the latter would be the noun phrase). When the NEBOnE plugin is installed, actions concerning the ontology are implemented on the RHS of JAPE rules, such as adding or deleting new classes, instances, subclasses, properties and so on.

If an item is selected for addition to the ontology as a new class, NEBOnE first checks to see whether the item in question already exists in the ontology: if it already exists in the place where it is scheduled to be added, NEBOnE will do nothing. If the item exists as a class elsewhere in the ontology, NEBOnE will add the new class (because it supports multiple inheritance). If the requested parent class and subclass both exist and are class names, NEBOnE will make the second a subclass of the first and print a message. If either is already an instance, or the parent class does not exist yet, NEBOnE will print a warning message.

Similarly for an instance, if it exists elsewhere as an instance, NEBOnE will add the new instance but generate a notification message. If the item already exists as a class, and an instance of the same name is to be added, or vice versa, then NEBOnE will not generate the new instance/class and will produce a warning message. Thus NEBOnE ensures consistency in the ontology, avoiding the need to run a checker after the ontology has been modified. A user can of course choose to ignore any potential inconsistencies, by checking the generated messages and then manually adding any offending items or making other changes to the ontology.

Unlike CLOnE, NEBOnE's functions will create classes and superclasses as required in order to accommodate instances and subclasses, respectively; it does not require every class to be explicitly created before it is used. The NEBOnE library does, however, reject function calls that would otherwise try to create an instance with the same name as an existing class, or the other way round.

## 3 Related work

Lexico-syntactic pattern-based ontology population has proved to be reasonably successful for a variety of tasks [13]. The idea of acquiring semantic information from texts dates back to the early 1960s with Harris' *distributional hypothesis* [14] and Hirschman and Sager's work in the 1970s [15], which focused on determining sets of sublanguage-specific word classes using syntactic patterns from domain-specific corpora. A detailed description and comparison of lexical and syntactic pattern matching can be found in [16], In particular, research in this area has been used in specific domains such as medicine, where a relatively small number of syntactic structures is often found, for example in patient reports. Here the structures are also quite simple, with short and relatively unambiguous sentences typically found: this makes syntactic pattern matching much easier.

Text2Onto [17] performs synonym extraction on the basis of patterns. It combines machine learning approaches with basic linguistic processing such as tokenisation or lemmatisation and shallow parsing. Since like SPRAT it is based on the GATE framework, it offers flexibility in the choice of algorithms to be applied. Compared with SPRAT, it has a smaller number of lexico-syntactic patterns. On the other hand, it applies additional statistical clustering and parsing for relation extraction All in all, this leads to more data, but not necessarily to an improvement of the resulting ontology in terms of precision.

We also took inspiration from some currently unpublished research carried out at DFKI in the Musing project[3], which looks at deriving T-Box Relations from unstructured texts in German. In this work, attention is focused primarily on deriving relations between parts of German compound nouns, but we can make use of similar restrictions. For example, in their work they might derive from the compound noun "bank manager" that there is a property "has manager" belonging to "bank", and that a "bank manager" is a subclass of "manager".

Within the range of activities required for ontology learning, SPRAT covers a number of intermediate stages in the process of ontology acquisition, namely term recognition and relation extraction. In the initial acquisition stage, it will recognise terms from the corpus only if they participate in any of the patterns. This guarantees termhood only up to a certain extent. Further term filtering results in improved precision. For relation extraction, SPRAT does not make use

[3] http://www.musing.eu

of a parser. There are many applications that make use of syntactic dependencies e.g. [18, 19]. Our approach differs from this in that our patterns are defined at low levels of syntactic constituency, such as noun phrases, and by means of finite state transducers. Identifying and engineering on the basis of the linguistic building blocks that are relevant for each ontology editing task eliminates the need for a parser. Patterns are encoded locally, i.e. not embedded into a syntactic structure., but bottom-up created by combinations of finite-state patterns. Pattern variations can therefore be easily encoded. This bottom-up approach is much faster and less error-prone than a parser, because it robustly identifies syntactic building blocks rather than complete syntactic parses. Our approach is more in line with the ontology bootstrapping approach advocated in [20].

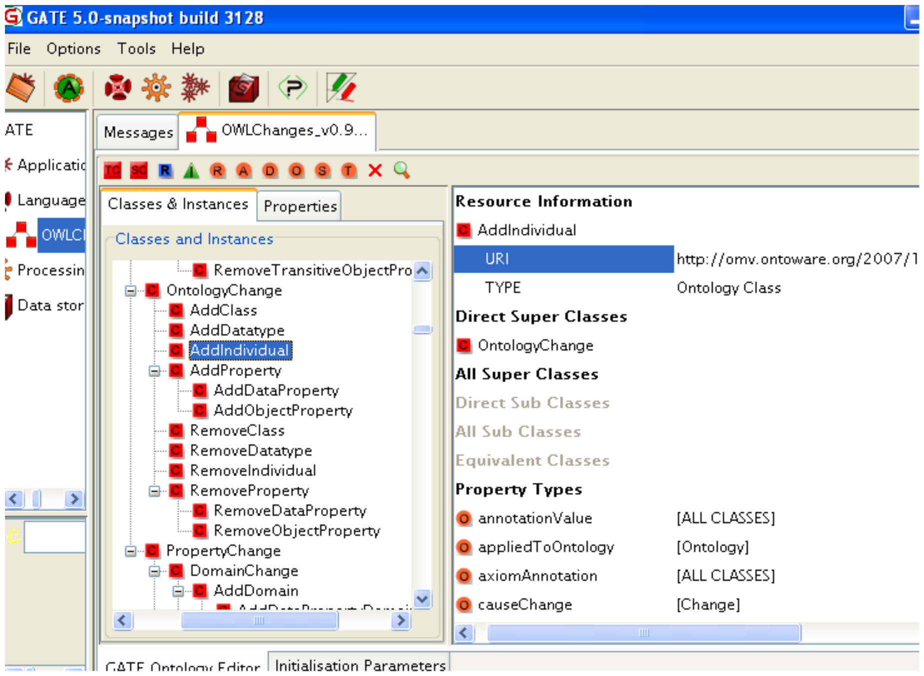
## 4 Ontology change management in GATE

The bottom-up approach to ontology change embodied by applications such as SPRAT is only half the story. The ontology development lifecycle also requires a change log management system which allows the managing of changes made by users in a collaborative environment [21]. This is necessary because different people, who may be experts in their domain, sometimes end up making changes to the same ontology, often at distributed locations. Where it is not possible to have access to a shared repository, other people wishing to make changes to the same ontology have to wait for others to finish their tasks. Changes made to an ontology may involve additions, deletions and modifications [22].

The basic idea behind producing a change log is to allow people to share their changes with others and thus collaborate on manipulating ontology data. This allows them to work independently but simultaneously on the same ontology. However, some ontologies can be extremely large and unwieldy, whereas the changes made to such ontologies may be only very minor. Instead of exchanging different versions of ontologies, it makes it easier to exchange change logs which can then be applied over the original ontology to get the modified version.

As an extension to the OMV (Ontology Metadata Vocabulary)[4], a taxonomy has been defined called *OWLChanges*, consisting of a concept in the ontology representing each different type of change possible. Figure 2 shows a snapshot of part of this ontology, loaded in GATE. This includes concepts such as *AddClass*, *RemoveClass*, *AddIndividual*, *RemoveIndividual* etc. In the case of the NeOn Toolkit change log, every change made to an ontology is recorded as an instance of the relevant concept in the *OWLChanges* ontology. Recording changes this way makes it possible to represent the final change log as a separate ontology where every instance refers to a change made in the ontology. In addition to the information about changes made by users, the change log contains other useful information such as who made those changes, when and in which order etc. Since the change log is a valid OWL ontology, it can be loaded as well as queried independently.

[4] http://ontoware.org/projects/omv/



Classes and Instances

```
    RemoveTransitiveObjectPro
  OntologyChange
    AddClass
    AddDatatype
    AddIndividual
    AddProperty
      AddDataProperty
      AddObjectProperty
    RemoveClass
    RemoveDatatype
    RemoveIndividual
    RemoveProperty
      RemoveDataProperty
      RemoveObjectProperty
  PropertyChange
    DomainChange
      AddDomain
```

Resource Information

AddIndividual

| | |
|---|---|
| URI | http://omv.ontoware.org/2007/1 |
| TYPE | Ontology Class |

Direct Super Classes

OntologyChange

All Super Classes

Direct Sub Classes

All Sub Classes

Equivalent Classes

Property Types

| | |
|---|---|
| annotationValue | [ALL CLASSES] |
| appliedToOntology | [Ontology] |
| axiomAnnotation | [ALL CLASSES] |
| causeChange | [Change] |

**Fig. 2.** Part of the OWLChanges ontology loaded in GATE

GATE has several different plugins useful for carrying out different types of information extraction tasks. It also has its own ontology API, which can be used together with these resources to take the maximum advantage of the combination. There are several applications that use these resources and manipulate ontologies (automatically) using the ontology API. We implemented GATE Ontology Services (GOS) to allow people to connect to a central repository and manage ontologies centrally. It has been implemented in such way that it can be used with other resources available in GATE.

GOS supports storing ontologies in a shared repository on a remote server. GOS is based on OWLIM [23], which is a high performance semantic repository developed in Java. It is packaged as a Storage and Inference layer (SAIL) for the Sesame[5] RDF database. It has its own published API that a user can use to make changes to the ontologies stored on the server. Software clients such as the *NeonOntologyServiceClient* allow users to connect to this server, upload new ontologies or use existing ones and manipulate data using the service methods published by the GOS. Similar to the NeOn Toolkit, GOS also produces a change log that describes changes made by the various users.

[5] http://www.openrdf.org/

When making changes to the ontology, users contribute to the change log maintained on the server. The GOS maintains only one instance of change log per repository and accumulates changes made by different users in the same change log. Users wishing to download the entire change log can do so by selecting one of the options provided in the *NeonOntologyServiceClient*. When users download a change log, it not only contains changes made by them but also the changes that took place ever since the server was started.

In its current version, GOS inherits Sesame's repository management system which allows giving different rights to different users on different repositories. In other words if proper rights are assigned, multiple users can connect to the same repository from different locations and make changes. It is up to the system using OWLIM/Sesame as backend to utilise this functionality and restrict multiple users from editing the same ontology at the same time. In GOS, currently only one user is allowed to modify the ontology at a time. Other users are given read-only access if they connect to a repository which is already being edited by another user. However, since the different repositories are stored on the same central server, different users connecting to the same repository one after the other see the latest modified version. If simultaneous access is needed, one can carefully distribute or make copies of the same ontology across different repositories and ask users to connect to a different copy. Users wishing to make changes can then contribute by adding new instances or deleting existing ones in their copy. GOS does not allow making changes to the implicit resources (imported ontologies). This helps in restricting users from making changes to the basic taxonomy. Since making changes produces change logs, these change logs can be investigated for debugging purposes. More information on GOS, how to access it, its methods and the change log can be found in [24].

One of the problems with the original version of the GOS change log is that it restricted users to use it only within GOS and was not directly compatible with the NeOn Toolkit. Change logs produced by GOS could only be interpreted by GOS. The same was true for NeOn Toolkit change logs which could not be interpreted by GOS. This meant that users of GATE who modified an ontology could not publish their changes to the NeOn Toolkit so that toolkit users having access to the same ontology could make use of them. Similarly if a toolkit user modified an ontology, the change log was not applicable in GATE.

It is important that the GOS change log is not restricted to use only with GATE, otherwise the modified ontology is only useful within GATE itself and cannot be then used inside other tools, architectures and ontology editors. We therefore provide an implementation in the GOS that not only understands but also produces the change logs that are compatible with the NeOn Toolkit. In other words, it bridges the gap between the GOS and the NeOn Toolkit that will allow people to use and transport change logs (instead of the entire ontologies) across the two systems. For example, a typical scenario involves an ontology originating from an organisation such as FAO, a part of which is then modified in the NeOn Toolkit using the loosely coupled SPRAT plugin. Since SPRAT relies on the GATE architecture, the change log is produced by GATE as part of the ontology modification process. In order for the new version of the ontology to be then recoupled with the existing original ontology, and for further tools to be used on it, the change log must be readable also by the NeOn Toolkit so that the changes can be applied. Figure 3 shows the interaction between a GATE application (SPRAT), the change log created and the Neon Toolkit.

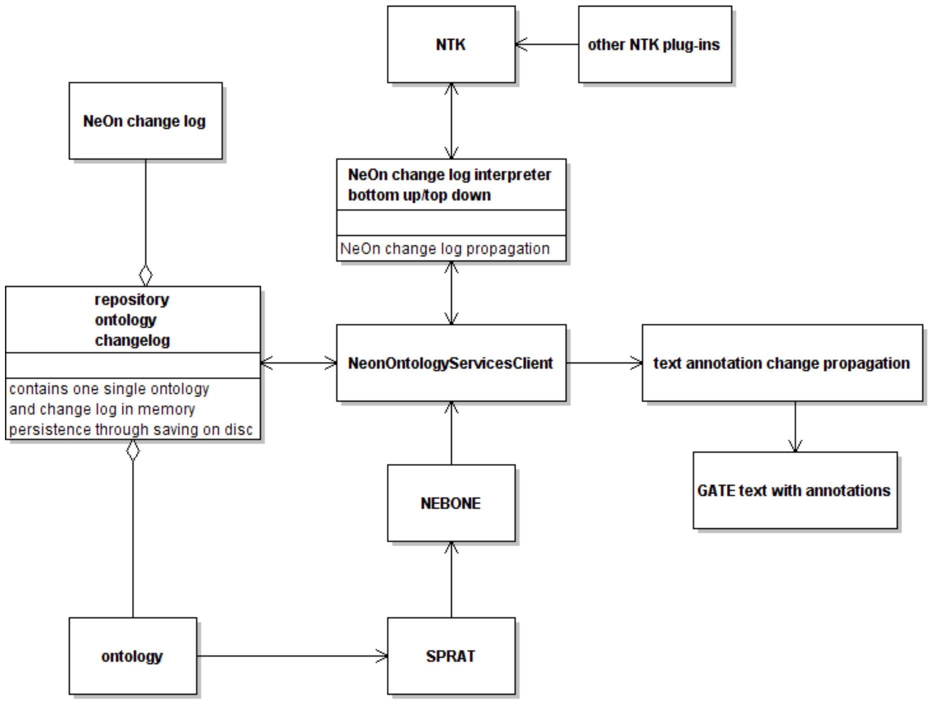

NTK
other NTK plug-ins
NeOn change log
NeOn change log interpreter bottom up/top down
NeOn change log propagation
repository ontology changelog
contains one single ontology and change log in memory persistence through saving on disc
NeonOntologyServicesClient
text annotation change propagation
GATE text with annotations
NEBONE
ontology
SPRAT

**Fig. 3.** Interaction between GATE, the change logs and the NeOn toolkit

The system works in both directions. Given this setup, users can choose to work using a system of their choice (NeOn Toolkit or GATE) and produce a change log that can be interpreted by both the toolkit and GOS to bring the ontologies in the same state.

The *NeonOntologyServiceClient* uses the graphical interface of the Gate Ontology Editor to allow users to create and populate new and existing ontologies. The same ontology editor is also used by a standalone resource known as *OWLIMOntologyLR* [6] which is a part of the core GATE system.

In a scenario where different people are contributing their efforts into the final outcome, it is very important to know what was contributed by whom. The client adds information about the authors along with the timestamps to record when

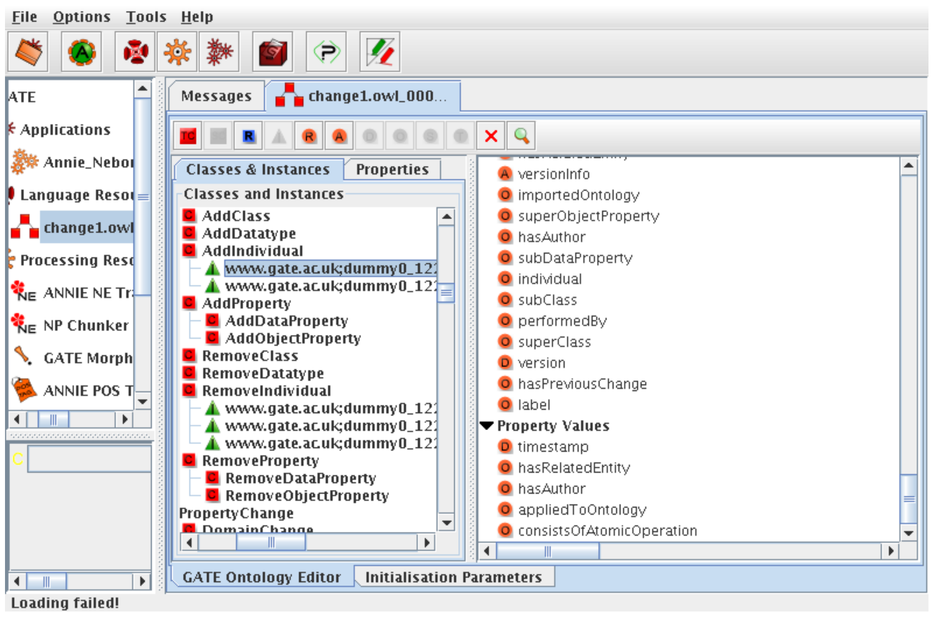
[6] http://gate.ac.uk/sale/tao/splitch10.html#x12-34800010.3

the changes occurred. Change logs are produced as valid RDF/XML documents where every change made to an ontology is an instance of some appropriate class.

## 4.1 Recording and interpreting changes

The change ontology, as explained earlier, has a number of concepts that define every possible change in the ontology. For example, whenever a new class is added, an instance of *AddClass* is created and added to the change log. This instance has several properties such as when the change occurred, who made the change, which ontology this change belongs to, the URI(s) of the affected resource(s) and so on.

Every change in the ontology can be classified as either an *Addition* or a *Removal*. An instance of a concept called *ChangeSpecification* is created for every Addition or Removal made to the ontology. This instance is associated with the axiom that provides more details about the change itself. Since the changes are recorded in the ontology, it is difficult to say which change occurred first. However, in order to apply the changes, it is very important to know the order in which changes occurred. To solve this problem, every change is associated with the change that took place just before the latest change, using a property called *hasPreviousChange*. Iterating over the values of these properties helps to identify the correct order in which these changes occurred.



Loading failed!

**Fig. 4.** Change log saved after modifications to the initial ontology

Because every change log is an ontology, it is possible to store each one under a separate ontology repository in GOS. As explained above, different statements are added to this repository for registering different changes in the ontology. GOS allows the exporting of ontologies in different formats. Thus the change log can be exported in formats such as RDF/XML, NTriples, N3 and Turtle. By default, the change logs are exported as RDF/XML. Having exported change logs in one of the above formats, one can easily load them in any ontology editor to see the change log as a separate ontology. Figure 4 depicts a snapshot of the change log in the GATE Ontology Editor. It shows details such as the timestamp, username, affected resource etc. of the first change made to the ontology. Having done this, the next task is to load such change logs back and apply them over to ontologies. GOS has an option that allows users to perform this task.

### 4.2 Compatibility Testing

The development of the GOS change logs presented here is very new and needs thorough compatibility testing with both the GOS and the NeOn Toolkit systems. At least four tests need to be carried out to test the system. One of these involves producing a change log from GOS and checking if it is a valid RDF/XML. Since all the statements are added to a separate ontology repository, adding an invalid statement would cause immediate complaint. Since they do not produce complaint, we know that the ontologies (change logs) are guaranteed to be valid RDF/XMLs. The second test is to produce a change log from GOS and apply it back in GOS. The test is successful if the changes registered in the change log can be applied successfully in the correct order. Our system has successfully passed this test. The third test is to take a change log produced from the NeOn Toolkit and apply it over to the ontology in GOS. Finally, the fourth test is to take a change log produced from GOS and apply it over to the ontology in NeOn Toolkit. The last two tests are yet to be carried out and form part of the work planned for the coming months.

## 5 Conclusions

In this paper we have described the implementation of the approach to modelling some of the dynamics of (semantic) metadata, providing mutual support for ontology lifecycle development and an NLP toolkit. We have provided a tool for people to derive new semantic information from unstructured text and represent this information ontologically so that it may then be used for further processing. The importance of this work is the interoperability achieved between the ontology editing and natural language processing architectures. Users of both tools may decide to modify the ontologies in question or to update the source data or facts derived from them. By creating mechanisms for the interchange and propagation of information in this way, including valuable change log data and support for multiple users working individually or collaboratively in a potentially distributed environment with networked ontologies, we have released an important bottleneck.

**Acknowledgements.** This research was partially supported by the EU Sixth Framework Program project NeOn (IST-2005-027595).

## References

1. Maynard, D., Peters, W., d'Aquin, M., Sabou, M.: Change management for metadata evolution. In: ESWC International Workshop on Ontology Dynamics (IWOD), Innsbruck, Austria (June 2007)
2. Maynard, D., Li, Y., Peters, W.: NLP Techniques for Term Extraction and Ontology Population. In Buitelaar, P., Cimiano, P., eds.: Bridging the Gap between Text and Knowledge - Selected Contributions to Ontology Learning and Population from Text. IOS Press (2008)
3. Bontcheva, K., Davies, J., Duke, A., Glover, T., Kings, N., Thurlow, I.: Semantic Information Access. In Davies, J., Studer, R., Warren, P., eds.: Semantic Web Technologies. John Wiley and Sons (2006)
4. Noy, N., Sintek, M., Decker, S., Crubézy, M., Fergerson, R., Musen, M.: Creating Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems **16**(2) (2001) 60–71
5. Fellbaum, C., ed.: WordNet - An Electronic Lexical Database. MIT Press (1998)
6. Schuler, K.K.: VerbNet: A broad-coverage, comprehensive verb lexicon. PhD thesis, University of Pennsylvania (2005)
7. Aswani, N., Tablan, V., Bontcheva, K., Cunningham, H.: Indexing and Querying Linguistic Metadata and Document Content. In: Proceedings of Fifth International Conference on Recent Advances in Natural Language Processing (RANLP2005), Borovets, Bulgaria (2005)
8. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Conference on Computational Linguistics (COLING'92), Nantes, France, Association for Computational Linguistics (1992)
9. de Cea, G.A., Gómez-Pérez, A., Ponsoda, E.M., Suárez-Figueroa, M.C.: Natural language-based approach for helping in the reuse of ontology design patterns. In: Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management Knowledge Patterns (EKAW 2008), Acitrezza, Italy (September 2008)
10. Maynard, D., Funk, A., Peters, W.: SPRAT: a tool for automatic semantic pattern-based ontology population. In: International Conference for Digital Libraries and the Semantic Web (submitted), Trento, Italy (2009)
11. Cunningham, H., Maynard, D., Tablan, V.: JAPE: a Java Annotation Patterns Engine (Second Edition). Research Memorandum CS–00–10, Department of Computer Science, University of Sheffield (November 2000)
12. Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Davis, B., Handschuh, S.: CLOnE: Controlled Language for Ontology Editing. In: Proceedings of the 6th International Semantic Web Conference (ISWC 2007), Busan, Korea (November 2007)
13. Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A., Shaked, T., Soderland, S., Weld, D.S., Yates, A.: Web-scale Information Extraction in KnowItAll. In: Proceedings of WWW-2004. (2004) `http://www.cs.washington.edu/research/knowitall/papers/www-paper.pdf`.
14. Harris, Z.: Mathematical Structures of Language. Wiley (Interscience), New York (1968)

15. Hirschman, L., Grishman, R., Sager, N.: Grammatically based automatic word class formation. Information Processing and Retrieval **11** (1975) 39–57
16. Maynard, D.G.: Term Recognition Using Combined Knowledge Sources. PhD thesis, Manchester Metropolitan University, UK (2000)
17. Cimiano, P., Voelker, J.: Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In: Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB), Alicante, Spain (2005)
18. Cimiano, P., Hartung, M., Ratsch, E.: Learning the appropriate generalization level for relations extracted from the Genia corpus. In: Proc. of the 5th Language Resources and Evaluation Conference (LREC). (2006)
19. Gamallo, P., Gonzalez, M., Agustini, A., Lopes, G., de Lima, V.: Mapping syntactic dependencies onto semantic relations. In: Proc. of the ECAI Workshop on Machine Learning and Natural Language Processing for Ontology Engineering. (2006)
20. Maedche, A.: Ontology Learning for the Semantic Web. Kluwer Academic Publishers, Amsterdam (2002)
21. Palma, R., Haase, P., Ji, Q.: Change management to support collaborative workflows. Technical Report D1.3.2, NeOn Project Deliverable (2009)
22. Maynard, D., Peters, W., D'Aquin, M., Sabou, M., Aswani, N.: Dynamics of metadata. Technical Report D1.5.1, NeOn Project Deliverable (2007)
23. Kiryakov, A.: OWLIM: balancing between scalable repository and light-weight reasoner. In: Proc. of WWW2006, Edinburgh, Scotland (2006)
24. Maynard, D., Peters, W., Angeletou, S., D'Aquin, M.: Implementation of metadata evolution. Technical Report D1.5.2, NeOn Project Deliverable (2008)