D6.6.4 Integration of GATE with KAON

Niraj Aswani (University of Sheffield)
Johanna Voelker (University of Karlsruhe)
Stephan Bloehdorn (University of Karlsruhe)
Kalina Bontcheva (University of Sheffield)

Abstract.
EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D6.6.4 (WP6)
This deliverable is a part of a series of bottom-up deliverables D6.6.x on bilateral integration activities and provides information on the integration of GATE and KAON2 tools.
Keyword list: KAON2, Text2Onto, GATE, bottom-up integration

Document Id.	SEKT/2005/D6.6.4/v0.2
Project	SEKT EU-IST-2003-506826
Date	January 23, 2006
Distribution	public

SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

British Telecommunications plc.
Orion 5/12, Adastral Park
Ipswich IP5 3RE, UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

Jozef Stefan Institute
Jamova 39
1000 Ljubljana, Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP, UK
Tel: +44 114 222 1891, Fax: +44 114 222 1810
Contact person: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

Intelligent Software Components S.A.
Pedro de Valdivia, 10
28006 Madrid, Spain
Tel: +34 913 349 797, Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

Ontoprise GmbH
Amalienbadstr. 36
76227 Karlsruhe, Germany
Tel: +49 721 50980912, Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Vrije Universiteit Amsterdam (VUA)
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam, The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Siemens Business Services GmbH & Co. OHG
Otto-Hahn-Ring 6
81739 Munich, Germany
Tel: +49 89 636 40 225, Fax: +49 89 636 40 233
Contact person: Dirk Ramhorst
E-mail: dirk.ramhorst@siemens.com

Empolis GmbH
Europaallee 10
67657 Kaiserslautern, Germany
Tel: +49 631 303 5540, Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

University of Karlsruhe, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe, Germany
Tel: +49 721 608 6592, Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Innsbruck
Institute of Computer Science
Technikerstraße 13
6020 Innsbruck, Austria
Tel: +43 512 507 6475, Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Kea-pro GmbH
Tal
6464 Springen, Switzerland
Tel: +41 41 879 00, Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

Sirma Group Corp., Ontotext Lab
135 Tsarigradsko Shose
Sofia 1784, Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Universitat Autonoma de Barcelona
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vallès)
Barcelona, Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Changes

Version	Date	Author	Changes
0.1	26.10.05	Stephan Bloehdorn	document setup
0.2	13.12.05	Niraj Aswani	KAON2 - GATE Integration
0.3	16.12.05	Kalina Bontcheva	added text to future integration and conclusion
0.4	16.01.06	Niraj Aswani	Final changes

Executive Summary

This deliverable is a part of a series of bottom-up deliverables D6.6.x on bilateral integration activities and provides information on the integration of GATE and KAON2 tools.

GATE is comprised of an architecture, a free open source framework (or SDK) and graphical development environment. It is used for all sorts of language processing tasks, including Information Extraction in many languages. As part of the GATE-Text2Onto integration, various GATE components such as tokenizer, stemmer, and part-of-speech tagger, have been integrated/used for linguistic processing in the Text2Onto system.

Text2Onto is a framework for data-driven change discovery by incremental ontology learning. Text2Onto combines machine learning approaches with basic linguistic processing such as tokenization or lemmatization and shallow parsing. To perform linguistic processing it relies on GATE.

KAON2 is a complete infrastructure for managing OWL-DL and SWRL ontologies. It serves as the main ontology management infrastructure within SEKT and also as the backbone for ontology evolution functionalities within SEKT. At the same time, the GATE ontology support is aimed at providing an abstraction layer between the actual representation mechanism and the NLP modules making use of it.

The KAON2-GATE integration allows users to manage ontologies using KAON2 from GATE. This integration will enable the Ontology-Based IE tools (see deliverables D2.1.x) to access the SEKT ontologies via KAON2 and take this information into account when adding metadata to documents. In addition, deliverable D2.2.1 on Controlled Language Information Extraction has already benefited from this integration. The user is now able to create ontologies in natural language and save them within KAON2.

Contents

Chapter 1

Introduction

SEKT aims at developing technologies for Next Generation Knowledge Management that exploit complementary research fields like Machine Learning (ML), Human Language Technology (HLT) and Ontology and Metadata Management (OM). Specifically, SEKT develops software to:

- semi-automatically learn ontologies and extract metadata,
- provide knowledge access,
- to perform middleware functionalities for global integration.

SEKT combines these core technologies to achieve synergy effects, whereby each technology is a key part of (semi-) automatic ontology learning (i.e. discovery of ontology primitives both on the schema and on the instance level) and maintenance.

This report is part of the bottom-up integration work performed in workpackage (WP) 6 on, specifically task 6.6. In the following, we put the work of this deliverable in the context of the SEKT project and outline its content.

1.1 The SEKT Big Picture

Figure 1.1 gives an overview of the SEKT project. The SEKT core technologies can be found among the 'Research & Development' activities: WP1 explores approaches for Ontology Generation from a machine-learning point of view with a strong focus on text mining. The corresponding tool suite, TEXTGARDEN, is mainly developed within WP1. WP2 uses Human Language Technology for Metadata Generation, especially within the framework of the GATE suite. WP3 researches ontology management infrastructures including functionalities for learning, updating and evolving ontologies over time and develops the KAON infrasturcture further with focus on the KAON2 and TEXT2ONTO

components. This report is part of the work performed in workpackage (WP) 6 on Integration and is naturally related with the technical workpackages and their interaction.

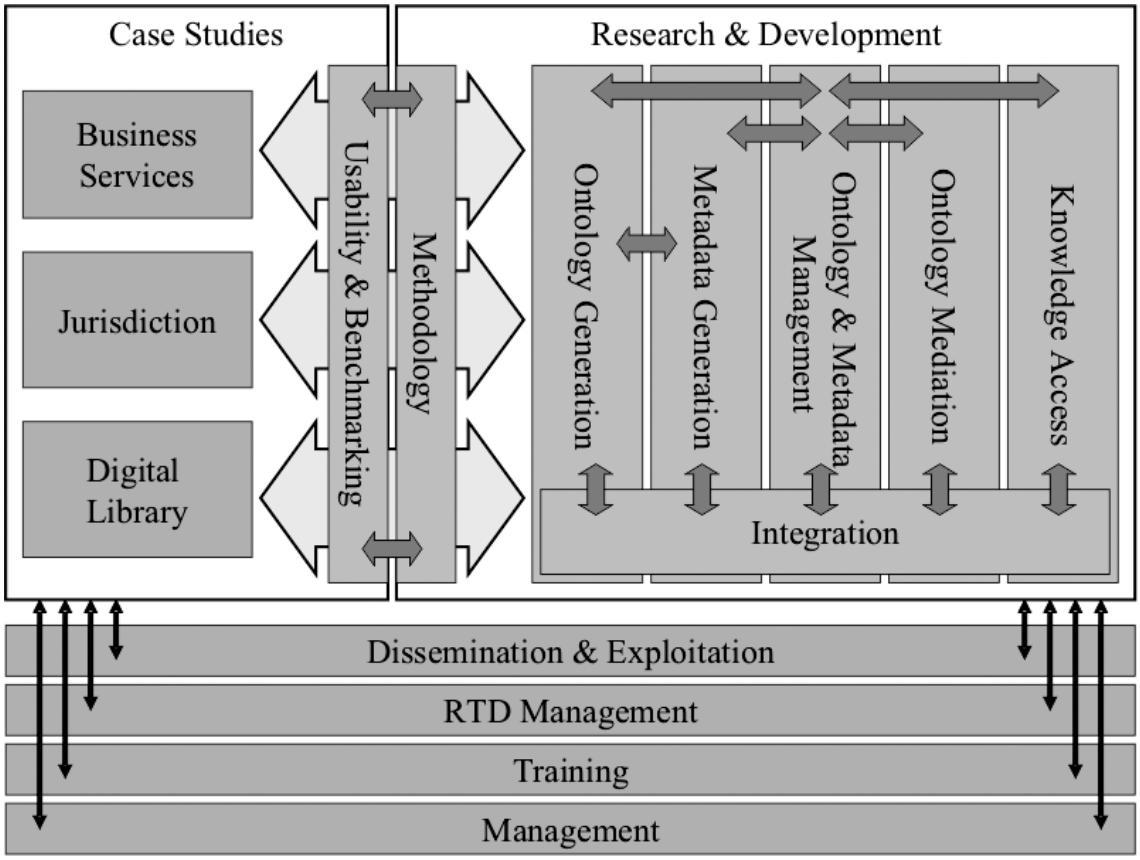

Figure 1.1: The SEKT Big Picture

1.2 Approaches to Integration in SEKT

Workpackage 6 aims at providing integration between the SEKT technical components in two ways (Figure 1.2). On the one hand it deals with a **top-down integration** approach via an Integration Plattform, on the other hand it deals with the complementary approach of **bottom-up integration** on a bilateral basis between the core research partners.

1.2.1 Top-Down Integration

In Workpackage 6, tasks 6.1 – 6.5 provide the overall architecture framework within which the SEKT technical components will be integrated. In the context of the work done so far it thus deals with the creation and extension of the the SEKT Integration Platform (SIP), which acts as the technical base for top-down integration.

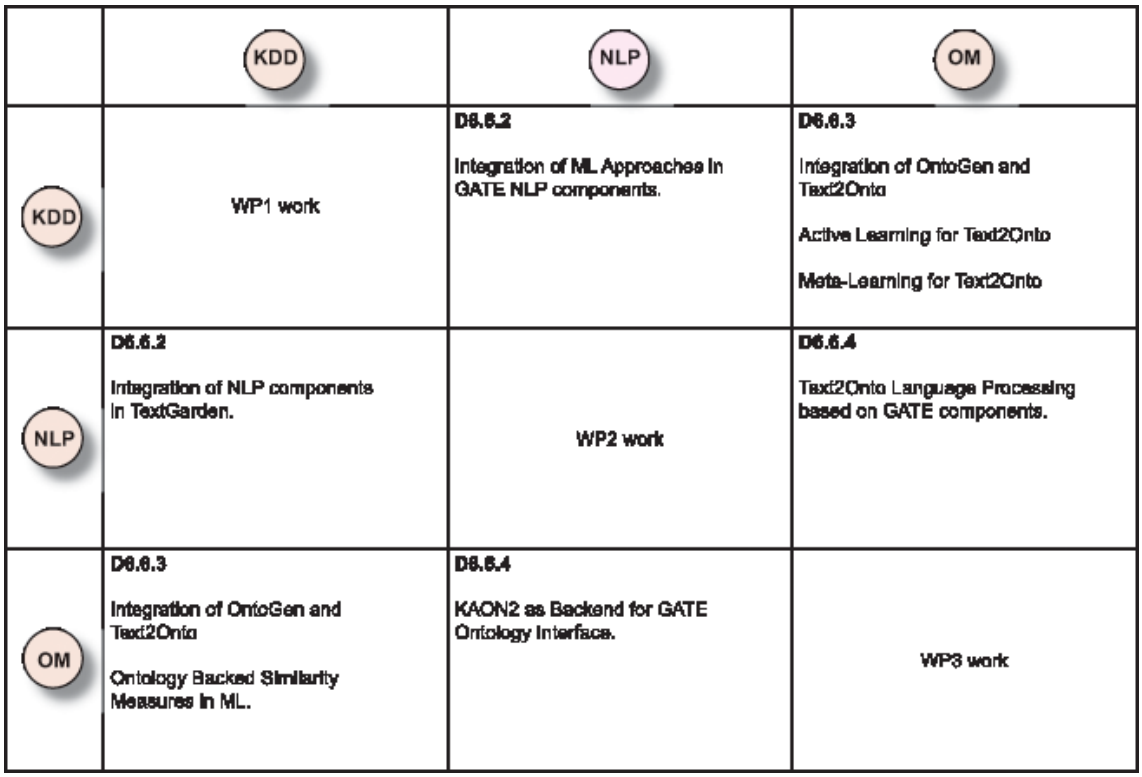

Figure 1.2: Integration Matrix

SIP offers modular extensibility for components by so-called "pipelets". Further information about the basic platform can be found e.g. in the SEKT deliverables D6.1 – D6.4. The deliverables D6.5.x describe the integration of components from the core workpackages into SIP, more specifically the integration of TEXTGARDEN, GATE and KAON (such as the OWL-DL inference engine KAON2) into SIP.

The top-down integration allows for bundling of integrated components in many different ways. This is particularly useful for bundling of SEKT technologies into enterprise solutions.

1.2.2 Bottom-Up Integration

In parallel, SEKT exploits a bottom-up integration where we integrate pairwise the core technologies developed in SEKT.

Task 6.6 which deals with bottom-up integration has been included after a first revision of results in task 6.5 at the end of year 1 of the SEKT project. Bottom-up integration explores the synergies which the SEKT core technologies offer for (semi-)automatic creation and maintenance of ontologies, or briefly ontology learning. It was felt necessary to better understand the low-level opportunities for integration — first to better under-

stand the top-down integration activities, second to emphasize more the research needed to combine SEKT technologies. The bottom-up approach complements the top-down strategy in different ways and mainly targets two things:

- Clarify the relationships between machine learning, human language technologies and ontology management.
- Coordinate the efforts for integration of the core technical components delivered in WP1, WP2 and WP3.

This explores the synergies on a bilateral basis between workpackages 1–3. The bottom-up integration task aims at light-weight proof-of-concept implementations which illustrate the potential and finally evaluate the potential of pairwise integration. This is particularly useful for further research prototyping.

After the initial status report in month 18 [BHS+05] which has reported on the overall bottom-up integration efforts from a birds-eye perspective, this deliverable is part of a series of bottom-up deliverables D6.6.x on bilateral integration activities, which aim to report in more technical detail on the bilateral integration efforts.

1.3 Outline

This section outlines structure of the deliverable and provides information on what information is included.

- The chapter 1 provides an overview of the SEKT project explaining various approaches adopted for the tools integrations.
- Chapter 2 provides generic descriptions of the (TextGarden, GATE and KAON2) tools and only focuses on the integration relevant details. Details on parts of the tools which are already SIP-enabled is provided followed by references to the relevant deliverables.
- Chapter 3 provides an overview of the Integration Scenarios. It provides short descriptions of completed and scheduled integration activties. The information on each entity such as how it relates to the overall SEKT goals, how could it be useful in the case studies and how it relates to the SIP integration is also provided. This also includes information from the end-users' perspective such as where to get the required resource(s) and how to use (from GUI and programmatically).
- More detailed information on prerequisites and how the tools were integrated is described under the chapter 4. Information on new resources (e.g. API) developed as part of the integration is also explained under this chapter.

- Chapter 5 describes the scheduled tool integrations and their expected outcomes.
- Finally, the deliverable is concluded with a short summary of the integration tasks and their outcomes.

Chapter 2

Description of the Individual Components for Integration

2.1 KAON Ontology Management

KAON is an open-source ontology management infrastructure with focus on scalable and efficient reasoning with ontologies and on learning/evolving ontologies. Two components of the KAON framework, Text2Onto and KAON2, are developed and used in the context of SEKT. These components are described in the following subsections.

2.1.1 Text2Onto

Text2Onto is a complete re-design and re-engineering of the TEXTTOONTO system, a tool suite for learning ontologies from textual data [MS01, Mae02]. Text2Onto is mainly developed from within workpackage 3 to provide functionalities in SEKT for Incremental Ontology Evolution, Usage Tracking for Ontologies and Metadata and especially for Data-driven Change Discovery [HV04].

Functionalities Text2Onto is aimed at learning/discovery of schema-level ontology primitives and instantiations from textual data by analyzing text documents based on linguistic background knowledge and statistical / machine learning approaches. Its main functionalities can be described as follows:

- Text2Onto represents the learned knowledge at a meta-level in the form of instantiated modelling primitives within a so called *Probabilistic Ontology Model (POM)*. Text2Onto thus remains independent of a concrete target language during learning and user supervision phase. At the same time it also translates the instantiated

primitives into any (reasonably expressive) knowledge representation formalism; especially into the OWL fragment supported by KAON2.

- user interaction is a core aspect of Text2Onto and the fact that the system calculates a confidence for each learned object allows to design sophisticated visualizations of the POM.
- KAON2 incorporates strategies for *data-driven change discovery* which allows for consecutive update and evolution of ontologies according to document corpus changes. Besides increasing efficiency in this way, it also allows a user to trace the evolution of the ontology with respect to the changes in the underlying corpus.

In addition to the core functionality of the Text2Onto described above, a graphical user interface featuring a corpus management component, a workflow editor, configuration dialogues for the algorithms as well as tabular and graph-based POMvisualizations has been developed. It will be available as an Eclipse[1] plug-in which could facilitate a smooth integration into ontology editors at a later development stage.

Technical Details Text2Onto is written entirely in Java and is thus platform independent. For interaction (import and export) with OWL ontologies, Text2Onto integrates with KAON2.

2.1.2 KAON2

KAON2[2] is a complete infrastructure for managing OWL-DL ontologies with DL-safe rules. It serves as the main ontology management infrastructure within SEKT and also as the backbone for ontology evolution functionalities within SEKT [HSV04a].

Functionalities Main functionalities of KAON2 are:

- a Java API for programmatic management of OWL-DL ontologies with DL-safe rules,
- a stand-alone server providing access to ontologies in a distributed manner,
- an inference engine for answering queries,
- a module for extracting ontology instances from relational databases (available soon),

[1] http://www.eclipse.org

[2] KAON2 is available as a precompiled binary distribution and is free of charge for research and academic purposes, see http://kaon2.semanticweb.org/

- a query interface for answering SPARQL[3] queries.

The API of KAON2 is capable of manipulating OWL-DL ontologies. Currently, the API can read ontologies in OWL XML Presentation Syntax and in OWL RDF Syntax. For reasoning, KAON2 supports the $\mathcal{SHIQ}(D)$ subset of OWL-DL (support for datatypes will be available soon). This includes all features of OWL-DL apart from nominals (also known as enumerated classes). Since nominals are not a part of OWL Lite, KAON2 supports all of OWL Lite. The API also provides no direct means for *creating* anonymous individuals. Indirectly, anonymous individuals are however possible by creating random URIs provided they're handled properly (i.e. excluded) in `owl:AllDifferent` and `owl:differentFrom` statements. By means of this mechanism, OWL ontologies that include anonymous individuals can still be processed by KAON2.

KAON2 also supports the so-called DL-safe subset [MSS04] of the Semantic Web Rule Language (SWRL). The restriction to the DL-subset has been chosen to make reasoning decidable. Contrary to most currently available DL reasoners, such as FaCT, RACER, DLP or Pellet, KAON2 does not implement the tableaux calculus. Rather, reasoning in KAON2 is implemented by novel algorithms which reduce a SHIQ(D) knowledge base to a disjunctive datalog program. For an overview of these algorithms, please refer to [HMS04b]. A detailed (and quite lengthy) technical presentation of all algorithms is given in [HMS04a].

Technical Details KAON2 is written entirely in Java and is thus platform independent.

2.2 Overview of GATE

GATE [CMBT02a] is an architecture, a framework and a development environment for LE (Language Engineering)[4]. As an *architecture*, it defines the organisation of an LE system and the assignment of responsibilities to different components, and ensures that the component interactions satisfy the system requirements. As a *framework*, it provides a reusable design for an LE software system and a set of prefabricated software building blocks that language engineers can use, extend and customise for their specific needs. As a *development environment*, it helps its users to minimise the time they spend building new LE systems or modifying existing ones, by aiding overall development and providing a debugging mechanism for new modules. Because GATE has a component-based model, this allows for easy coupling and decoupling of the processors, thereby facilitating comparison of alternative configurations of the system or different implementations of the same module (e.g., different parsers). The availability of tools for easy visualisation of data at each point during the development process aids immediate interpretation of the results.

[3] http://www.w3.org/TR/rdf-sparql-query/

[4] GATE is freely available for download from `http://gate.ac.uk`, under the LGPL license.

The GATE framework comprises a core library (analogous to a bus backplane) and a set of reusable LE modules. The framework implements the architecture and provides (amongst other things) facilities for processing and visualising resources, including representation, import and export of data.

The reusable modules provided with the backplane are able to perform basic language processing tasks such as POS tagging and semantic tagging. This eliminates the need for users to keep recreating the same kind of resources, and provides a good starting point for new applications. The modules are described in more detail in [CMBT02a].

Applications developed within GATE can be deployed outside its Graphical User Interface (GUI), using programmatic access via the GATE API (see `http://gate.ac.uk`). In addition, the reusable modules, the document and annotation model, and the visualisation components can all be used independently of the development environment.

GATE components may be implemented by a variety of programming languages and databases, but in each case they are represented to the system as a Java class. This class may simply call the underlying program or provide an access layer to a database; alternatively it may implement the whole component.

2.2.1 Ontology Support in GATE

As discussed earlier, the ontology support in GATE is aimed at providing an abstraction layer between the actual representation mechanism and the NLP modules making use of it. It consists of an in-memory data model for ontologies, an API providing access to that representation, a visual resource displaying the information, and input/output capabilities for accessing files containing ontologies using various standards.

Hierarchies of classes

The central role in the ontology data model is played by the class hierarchy, also known as a taxonomy. This consists of a set of classes linked by subClassOf and superClassOf relations. Classes have a name and a URI; in most cases the name is the local part of the URI, though this is not enforced. Class names within an ontology must be unique. Classes can also have comments which are used to explain their intended meaning.

All classes can have a set of super classes and a set of sub classes which are used to build the class hierarchy. The subClassOf and superClassOf relations are transitive and methods are provided by the API for calculating the transitive closure for each of these relations given a class. The transitive closure for the set of super classes for a given class is a set containing all the super classes of that class, as well as all the super classes of its direct super classes, and so on until no more super classes are found. This calculation is finite, the upper bound being the set of all the classes in the ontology. A class that has

no super classes is called a top/root class. An ontology can have several top/root classes. Although the GATE ontology API can deal with cycles in the hierarchy graph, these can cause problems for processes using the API and are probably denoting an error in the definition of the ontology. Care should be taken to avoid such situations.

Instances

Instances are objects that belong to classes. Like classes, they have a name and a URI and can have comments. One instance can belong to one or more classes and can have properties with values. There are API methods provided for getting all the instances that belong to a given class or the property values for a given instance.

Hierarchies of properties

The last part of the data model is made up of hierarchies of properties that can be associated with objects in the ontology. GATE defines three types of properties:

1. **Object Properties** are properties that are associated to an ontology instance and have an instance as value.
2. **Datatype Properties** are properties that are associated to an ontology instance and can have any Java object as value.
3. **Generic Properties** are associated to any ontology object, be it a class, an instance, or another property, and have any Java object as value.

From an OWL/RDF perspective, object and datatype properties are similar to the property types as defined by OWL variants while the generic properties are intended to model the properties defined by RDF.

Unlike some other representation models, in GATE, properties do not belong to classes, they are instead first-class citizens of the data model. The specification of the type of objects that properties apply to is done through the means of domains. A domain for a property is a set of ontology classes. The classes listed in the domain should be seen as a set of restrictions: for a property to be applicable to an instance, that instance needs to belong to all the classes in the property's domain. A property with a domain that is an empty set can apply to instances of any type (i.e. there are no restrictions given).

Similarly, the types of values that a property can take is restricted through the definition of a range. Object properties take instances as values so their ranges are also sets of classes; the manner in which the range restrictions are interpreted is similar to the case of the domain. Datatype and generic properties can have as value any Java object. Their ranges are sets of Java Classes (objects of type java.lang.Class) which means they can only have values that implement those classes. In most cases, such a range will either be empty (allowing any type of object) or contain a single Class value (the most specific Class from the class hierarchy that is appropriate). Having several classes that not in a hierarchical relation effectively blocks that property from accepting any value, while including several classes that hierarchically linked (derived from one another) is superfluous

as only the most specific one is significant. Of course a range can contain a set of objects representing interfaces, in which case non-singleton sets make sense.

Although the data-model and API permit the use of any kind of values for datatype and generic properties, if input/output is required using XML serialisations (such as XML-based OWL or RDF), then the allowed values are restricted by the set of types representable in those serialisations (usually the ones defined by the XML-Schema specification).

All properties can be marked as functional properties, which means that for a given instance in their domain, they can only take at most one value, i.e. they define a function in the algebraic sense. Properties inverse to functional properties are marked as inverse functional. Additionally, object properties can also be reflexive, symmetric, and transitive. If one likens ontology properties with algebraic relations, the semantics of these become apparent.

Similar to classes, properties can also be organised in hierarchies by means of establishing subPropertyOf and superPropertyOf relations. Methods are provided for obtaining the set of super- and sub-properties as well as the transitive closures of these sets.

The API provides methods for obtaining the names of the properties that are set for a given instance and for obtaining the sets of values for a given property and a given instance.

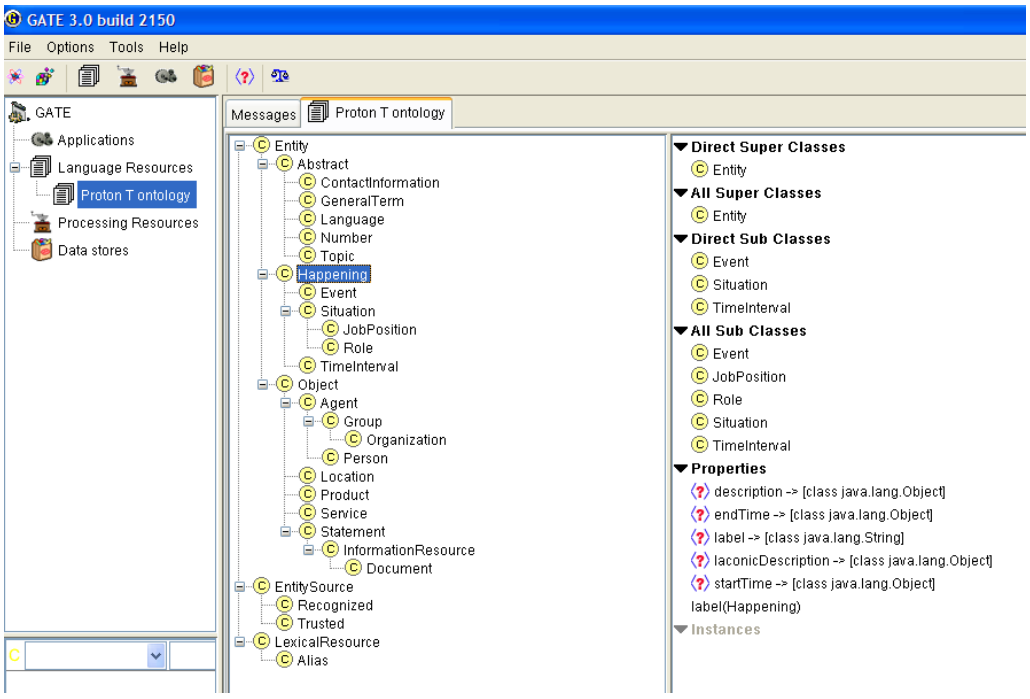

Figure 2.1: GATE's Ontology Viewer showing the Proton ontology open via KAON2

GATE's Ontology Viewer

The ontology support in GATE also includes a simple viewer that can be used to navigate an ontology and quickly inspect the information relating to any of the objects defined in it - classes, instances and their properties.

Figure 2.1 shows the GATE ontology viewer displaying a segment of the PROTON ontology[5] and the data associated with the class *Happening* which is currently selected. The ontology viewer is dynamic and updates the information displayed whenever the underlying ontology is changed through the API (See section 3.1.2).

[5] http://proton.semanticweb.org

Chapter 3

Overview on Integration Scenarios

The rationale behind the integration of KAON's ontology support within GATE comes from the well-established need for ontology access within NLP systems. The most common operations are accessing instances, properties, and concepts; sometimes adding new ones to the ontology; and loading and saving of the data. However, as ontology lookup is frequently performed as part of a complex NLP algorithm, e.g., co-reference resolution, it needs to be very effiecient. Therefore, a tight coupling between GATE and KAON is required, in order to enable individual algorithms to access ontologies in real time.

3.1 Using KAON2 from GATE

This section provides information from the end-users' perspective and explains how to use KAON2 Ontology module from GATE. Chapter 4 describes the integration of KAON2 in GATE in more details. In order to use KAON2 Ontology module from GATE, following instructions need be followed:

1. **Compiling with JDK 1.5** Since KAON2 is implemented in JDK 1.5, the user needs to recompile GATE with JDK 1.5.

2. **Installing the KAON2 plugin**

   Choose File/Manage CROEOLE plugins dialog. The select "Add a New CREOLE repository" button, and for URL type: http://gate.ac.uk/sale/gateplugins/kaon2. Finally, check the box "Load Now" (and optionally, "Load always"). This will enable you to create LRs of type KaonOntology. For further information on plugins in GATE, see: http://gate.ac.uk/gate/doc/plugins.html.

   If you want to use the GATE Ontology Viewer/Editor, please load the Ontology Tools plugin as well (see plugins page).

3. **Parameters** Two parameters are required in order to load an ontology using KAON2Ontology resource.

   (a) **defaultNameSpace** *www.gate.ac.uk/default* is the default value used for the defaultNameSpace parameter. While instantiating a particular ontology with KAON2Ontology, user needs to provide the ontology's relevant defaultNameSpace.

   (b) **ontologyURL** Ontology with the given URL is loaded in GATE. Leaving empty this paramter creates an empty ontology.

Instructions to use KAON2Ontology depend on whether it is being used from the GATE GUI or programmatically.

3.1.1 From GATE GUI

This section provides information on using the KAON2Ontology resource from the GATE GUI.

Instantiating KAON2 Ontology

The user needs to right click on Language Resources and select the KAON2 ontology option (Figure 3.1) from the new Language Resources menu. User then has to provide the appropriate parameters (3.2) and click OK to create an instance of the KAON2Ontology.

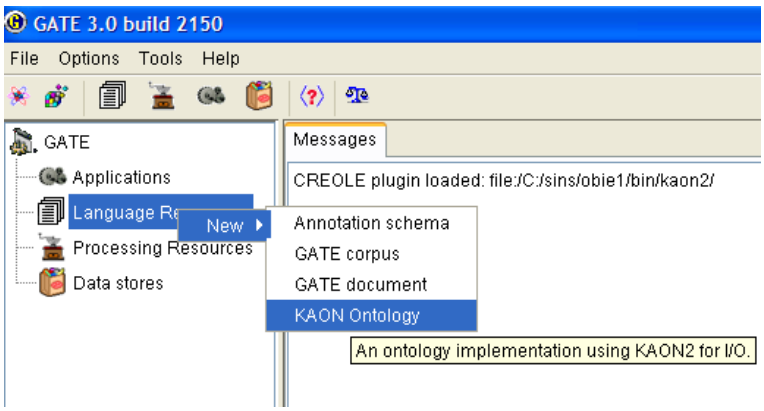

Figure 3.1: Instantiating ontology

Figure 2.1 shows the **protont** ontology[1] loaded in the GATE ontology editor.

[1] http://proton.semanticweb.org

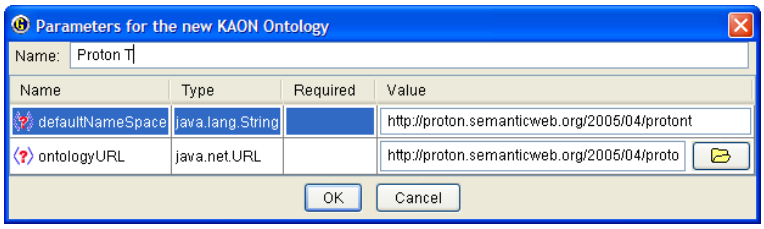

Figure 3.2: KAON2 ontology parameters

Saving an ontology

Ontologies can be saved in either OWL_XML or OWL_RDF format. In order to save, the user has to right click on the appropriate ontology instance and select the appropriate save option. Figure 3.3 displays a save menu with two options to save ontology in OWL_XML and OWL_RDF.

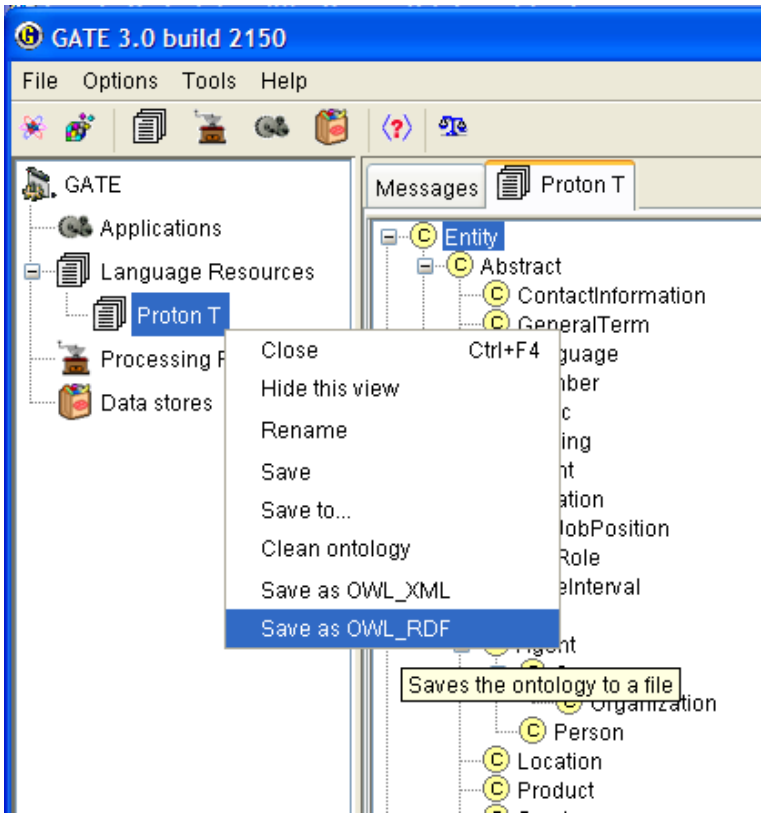

Figure 3.3: KAON2 ontology save options

3.1.2 Programmatically

KAON2Ontology serves as an I/O layer between the GATE ontology API and the KAON2 API. The following subsections explain how to use KAON2Ontology programmatically.

Loading KAON2

KAON2Ontology can be accessed from http://gate.ac.uk/sale/gateplugins/kaon2. In order to use it from GATE, user should register the above URL as a creole repository. The following code snippet shows how to initialize GATE and load KAON2Ontology.

```
// initialize GATE
Gate.init();

// create url
URL kaon2ontologyURL = new URL("http://gate.ac.uk/sale/gateplugins/kaon2");

// to register kaon2's creole.xml as a plugin in GATE
// the kaon2.jar (the KAON2 API) must be made available
// on the classpath
Gate.getCreoleRegister().registerDirectories(kaon2ontologyURL);
```

Instantiating KAON2 Ontology

Once the plugin of KAON2Ontology is loaded in GATE, one can instantiate KAON2Ontology as shown in the following code snippet. As mentioned earlier there are two paramters required. The following example shows how to load the PROTONT ontology in GATE.

```
// Two parameters
// 1. defaultNameSapce - the baseURI
String defaultNameSpace =
       "http://proton.semanticweb.org/2005/04/protont";

// 2. ontology URL
// In order to instantiate a new empty ontology,
// the ontologyURL parameter must be left blank
//  or set to null.
URL ontologyURL =
  new URL("http://proton.semanticweb.org/2005/04/protont");

// Create an instance of FeatureMap and add these parameters
FeatureMap features = Factory.newFeatureMap();
features.put("defaultNameSpace", defaultNameSpace);
features.put("ontologyURL", ontologyURL);

// finally create an instance of the kaon2ontology using
// the Factory.createResource
// the class that implements KAON2Ontology is KAON2OntologyImpl
// In order to instantite KAON2OntologyImpl, it
// invokes the KAON2OntologyImpl's init() method
// which internally invokes the load() method.
```

```
// load() method, retrieves all elements from the
// given ontology and map them to the GATE's local ontology
//  data structure
KAON2OntologyImpl ontology =
    (KAON2OntologyImpl) Factory.createResource(
    "gate.creole.ontology.kaon2.KAON2OntologyImpl",
    features);
```

Manipulating Ontology

New ontology elements (Concepts, Individuals and their properties) can be added or removed to and from the loaded ontology. In order to allow users to carry out these operations, KAON2OntologyImpl inherits all methods from its super class OntologyImpl. These methods are described below with examples.

The task is to add classes *Person* and *Man* where latter is the subclass of the former. There should be a class called *Liquid*. *Person* has an object property *drinks* where the range is specified as instances of the class *Liquid*. *John* is an instance of *Man*. *Water* is an instance of *Liquid*. Since *Person* can *Drink Liquid*; *John Drink*s *Water*. *Person* has a data property called *numberOfLegs*. *John* has 2 legs.

Below we demonstrate the use of GATE API to create a sample ontology for this description. For more details on the API, please refer to http://www.gate.ac.uk/gate/doc/javadoc/com/ontotext/gate/ontology/OntologyImpl.html.

```
// adding a new class Person
// name and comment
OClass person = ontology.createClass("Person","All humans");

// adding a new class Man
OClass man = ontology.createClass("Man", "Male Person");

// adding a new class Liquid
OClass liquid = ontology.createClass("Liquid",
                     "anything that is in liquid form");

// John is a person
OInstance john = ontology.addInstance("John", person);

// water is liquid
OInstance water = ontology.addInstance("Water", liquid);

// Man is a subclassof Person
person.addSubClass(man);

// Man drinks liquid
ontology.addObjectProperty("drinks",
           "man drinks liquid", man, liquid);

// john drinks liquid
john.addPropertyValue("drinks", water);

// every person has number of legs
ontology.addDatatypeProperty("numberOfLegs",
            "every person has some number of legs",
            person, Integer.class);
```

```
// john has two legs
john.addPropertyValue("numberOfLegs",new Integer(2));
```

Saving ontology

In order to save an ontology, the save() method should be called. It takes two parameters.

1. file should be an instance of the java.io.File. It specifies what name the ontology should be saved with.
2. file type - this can be either KAON2OntologyImpl.OWL_XML or KAON2OntologyImpl.OWL_RDF.

```
// calling the save method
// for example file should be saved in OWL_RDF
// with name proton.xml
File protont = new File("protont.rdf");
save(protont, KAON2OntologyImpl.OWL_RDF);
```

Chapter 4

Report on Current Tool Integration

4.1 Integration of GATE Components in Text2Onto

Text2Onto is a framework for ontology learning and ontology evolution developed as part of our efforts in WP3 [VS05a][VS05b]. This section gives a brief overview of a typical ontology learning process (section 4.1.1) and describes the use of different GATE components in Text2Onto (section 4.1.2).

4.1.1 Usage Scenario

A typical usage scenario for Text2Onto is depicted by figure 4.1. The user specifies a corpus, e.g. a collection of text, HTML or PDF documents, and starts the graphical workflow editor. The editor provides her with a list of algorithms which are available for the different ontology learning tasks, and assists her in setting up an appropriate workflow for the kind of ontology she wants to learn as well as to customize the individual ontology learning algorithms to be applied.

Once the ontology learning process is started, the corpus gets preprocessed by a GATE application (see section 4.1.2), before it is passed to the algorithm controller. In the following, depending on the configuration of the previously specified workflow, a sequence of ontology learning algorithms is applied to the corpus. Each algorithm starts by detecting changes in the corpus and updating the reference store accordingly. Finally, it returns a set of requests for changes regarding the POM, i.e. the *Preliminary Ontology Model* to its caller, which could be the algorithm controller, but also a more complex algorithm. After the process of ontology extraction is finished, the POM is presented to the user.

Since the POM unlike any concrete ontology is able to maintain thousands of conflicting modeling alternatives in parallel, an appropriate and concise visualization of the POM is of crucial importance for not overwhelming the user with too much information. Although several pre-defined filters such as a probability threshold will be available for

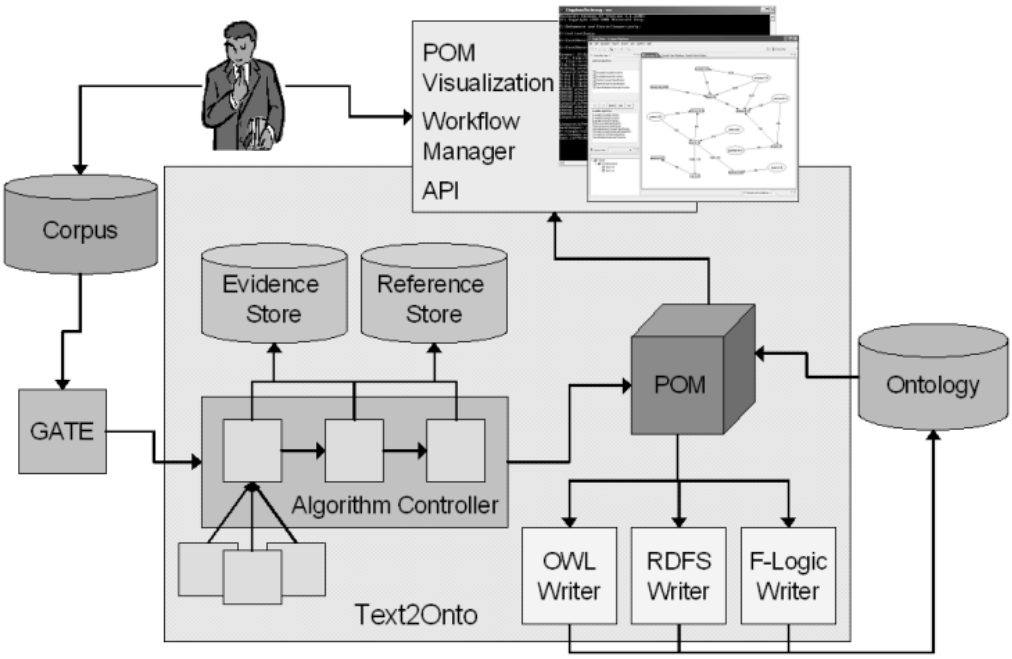

Figure 4.1: Text2Onto - Usage Scenario

pruning the POM, some user interaction might still be needed for transforming the POM into a high-quality ontology (see figure 4.2). After having finished her interaction with the POM, e.g. after adding or removing concepts, instances or relations, the user can select among various ontology writers, which are provided for translating the POM into different ontology representation languages.

4.1.2 Linguistic Analysis

Many existing ontology learning environments focus either on pure machine learning techniques [BNC00] or rely on linguistic analysis [BOS03, VNCN05] in order to extract ontologies from natural language text. Text2Onto combines machine learning approaches with basic linguistic processing such as tokenization or lemmatizing and shallow parsing. Since it is based on the GATE framework [CMBT02b] it is very flexible with respect to the set of linguistic algorithms used, i.e. the underlying GATE application can be freely configured by replacing existing algorithms or adding new ones such as a deep parser if required. Another benefit of using GATE is the seamless integration of JAPE [CMT00] which provides finite state transduction over annotations based on regular expressions.

Linguistic preprocessing in Text2Onto starts by tokenization and sentence splitting. The resulting annotation set serves as an input for a POS tagger which in the following assigns appropriate syntactic categories to all tokens. Finally, lemmatizing or stemming

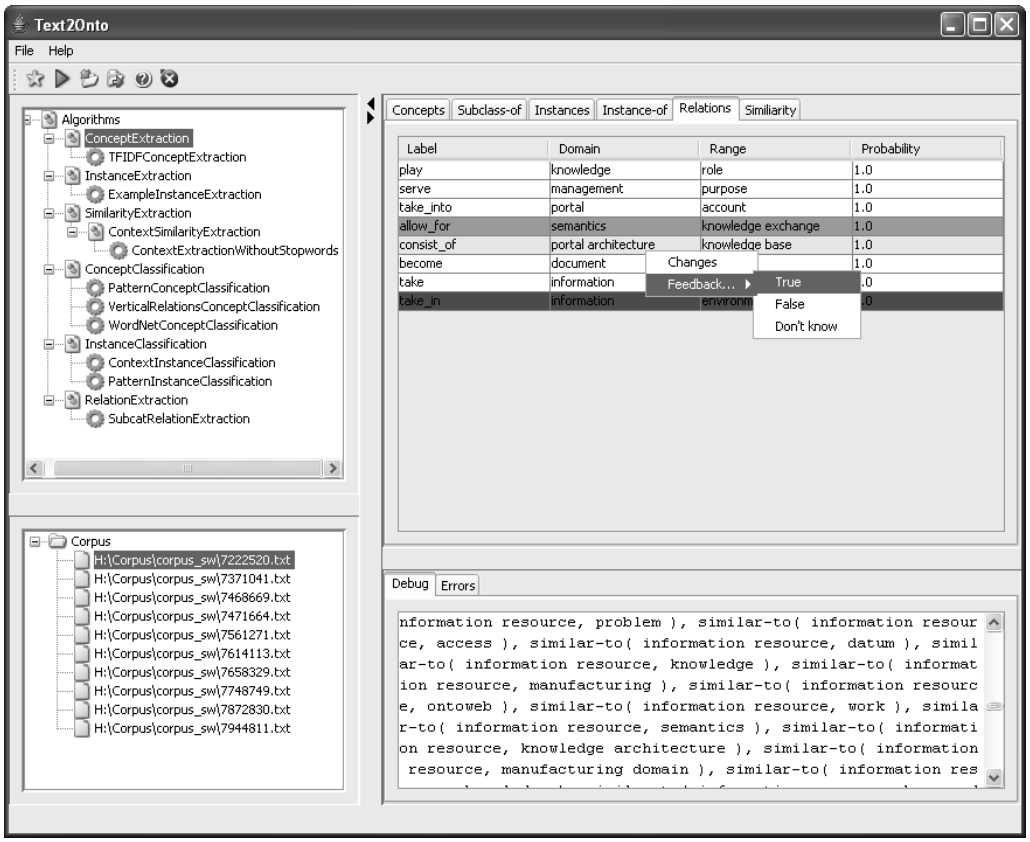

Figure 4.2: Text2Onto - User Feedback

(depending on the availability of the regarding processing components for the current language) is done by a morphological analyzer and a stemmer respectively.

After the basic linguistic preprocessing is done, a JAPE transducer is run over the annotated corpus in order to match a set of particular patterns required by the ontology learning algorithms. Whereas the left hand side of each JAPE pattern defines a regular expression over existing annotations, the right hand side describes the new annotations to be created (see figure 4.4). For Text2Onto we developed JAPE patterns for both shallow parsing and the identification of modeling primitives, e.g. concepts, instances and different types of relations (c.f. [Hea92]).

Since obviously, both types of patterns are language specific, different sets of patterns for shallow parsing and ontology extraction have to be defined for each language. Because of this and due to the fact that particular processing components for GATE have to be available for the regarding language, Text2Onto currently supports ontology learning from English and German texts. Fortunately, thanks to recent research efforts made in the SEKT project[1] GATE components for the linguistic analysis of various languages

[1] www.sekt-project.com

```
Rule: NounPhrase (
 (
  {Token.category == DT}
  {SpaceToken.kind == space}
 )?
 (
  ({Token.category == JJ}|{Token.category == VBG})
  {SpaceToken.kind == space}
 )*
 (
  (
   ({Token.category == NN} | {Token.category == NNS})
   {SpaceToken.kind == space}
  )*
  ({Token.category == NN} | {Token.category == NNS})
 ):np_head
):np
-->
:np.NounPhrase = { rule = "NounPhrase" },
:np_head.Head = { rule = "NounPhrase" }
```

Figure 4.3: JAPE pattern: Noun Phrase

such as Spanish [May05] have been made available recently. Since we want to provide full support for all of these languages in future releases of Text2Onto, we have already integrated some of these components, and we are currently working on the development of appropriate patterns for Spanish.

4.2 Integration of KAON2 within GATE

4.2.1 Loading and Saving Ontologies in KAON2 from GATE

KAON2 supports OWL DL and OWL Light. Since it requires JDK 1.5, it is implemented as an external plugin module and is available as a GATE Language Resource (*KAON2 Ontology*). KAON2OntologyImpl serves as a basic ontology I/O (Figure 4.5) layer between GATE and KAON2.

KAON2OntologyImpl implements two methods: load() and save(). Using KAON2 API, the method load(), accesses the ontology elements (i.e. concepts, properties, instances) and populates them in the GATE ontology data structure. Following actions are considered while transfering information into the GATE ontology data structures.

```
Rule: Hearst_1 (
 (NounPhrase1):superconcept
 (
  {Token.kind == punctuation}
 )?
 {SpaceToken.kind == space}
 {Token.string == "such"}
 {SpaceToken.kind == space}
 {Token.string == "as"}
 {SpaceToken.kind == space}
 (NounPhrasesAlternatives):subconcept
):hearst1
-->
:hearst1.SubclassOfRelation = { rule = "Hearst1" },
:subconcept.Domain = { rule = "Hearst1" },
:superconcept.Range = {rule = "Hearst1" }
```

Figure 4.4: JAPE pattern: Hearst

1. Create a new connection to KAON2 Server.
2. Open an ontology using KAON2 API.
3. Specifiy the Source URI.
4. Find out all root classes (classes with no super classes) using the KAON2 API. Each class needs to be converted into the GATE ontology class object (OClass) and to be added into the local data structure. For each class, all its subclasses are obtained and a hierarchy for all ontology classes is built. When retrieving information about an ontology element, all imported ontologies are also considered. The parameters needed for creating a new ontology class are its URI, local name (the value after '#' in URI), class label (the value of rdfs:label attribute), and comment (the value of rdfs:comment attribute).
5. Find out all properties (Object, Datatype and Annotation Properties) using KAON2 API. AnnotationProperties are converted into Generic properties. For example comment and label annotations are considered as generic properties in GATE.
6. Mark properties as either Funcational or Inverse Functional. For Datatype properties, the default value of Inverse Functional attribute is set to false.
7. Object properties can be Transitive, Symmetric, Functional or Inverse Functional.
8. Heirarchy is built to reflect the super- and sub-properties.

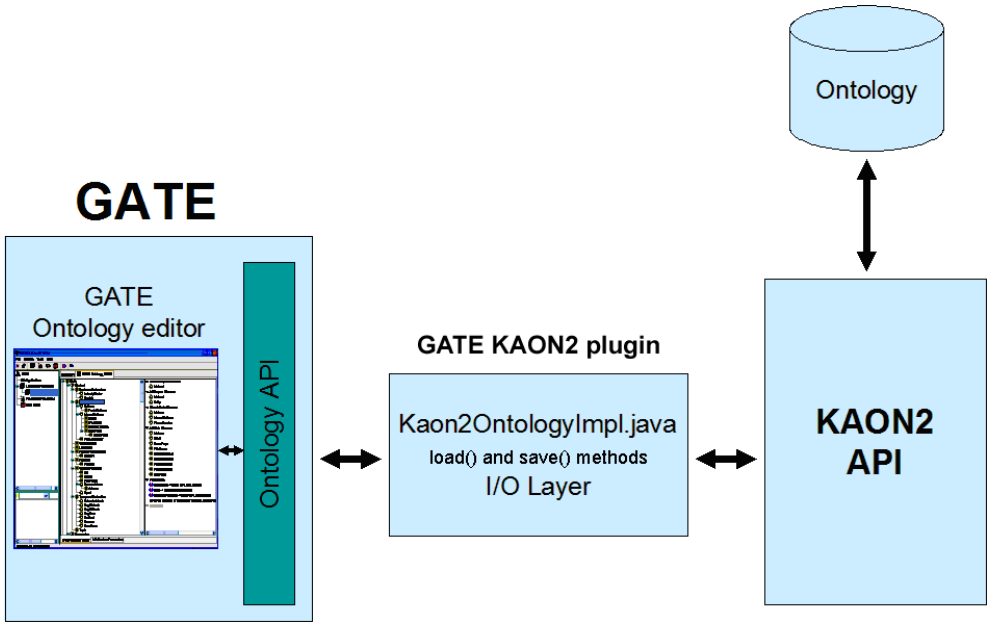

Figure 4.5: Instantiating ontology

9. Instances (Individuals in KAON2) need to be read and converted into Gate Instances.

10. For each Instance all its properties need to be read (Object and Datatype).

Ontologies can be saved in either OWL_XML or OWL_RDF format. Following actions are taken under the save() method.

1. A new connection to KAON2 server is created.

2. A new ontology is created using the source URI of the ontology.

3. For each class in the ontology, an equivalent OWL class is created using the KAON2 API.

4. Comments and Labels are added as AnnotationProperties using the KAON2 API.

5. For all Object and Datatype properties their equivalent KAON2 objects are created and their Domain and Range are specified. Any other property which is neither an Object nor Datatype property is converted in to the Annotation property.

6. The hierarchy is built for all properties in the ontology.

7. For each instance in the ontology an equivalent KAON2 Individual is created. Each Instance is then associated with its Class in the ontology.

8. Relevant properties are created and associated with their respective instances.

9. Based on the provided parameters, ontology is saved in either OWL_XML or OWL_RDF format. The encoding to save RDF is set to ISO-8859-1.

4.3 The Utility API in GATE for Interfacing with KAON2

KAON2 is implemented in JDK 1.5 and the binary distribution of the KAON2 contains several examples. These examples are very well documented which illustrate how to use different API functionalities. The current integration does not support reasoning functionalities and provides basic functionalities such as manipulation of ontology elements (add, remove and retrieve ontology elements). At present, this is sufficient for the requirements of NLP applications using ontologies in GATE.

As part of the integration process, a class called gate.creole.ontology.kaon2.KAON2Utils is implemented, which provides various static methods to manipulate ontologies using KAON2 API. These methods are used in gate.creole.ontology.kaon2.KAON2OntologyImpl to retrieve and save back the information in the ontology. These methods hide the code required to access particular information using KAON2 API. Below we provide some examples to demonstrate basic functionalities followed by the use of the static methods available in KAON2Utils.

Opening a new ontology

```
// the first step is to create a new connection
// using KAON2Manager.  This connection is used later
// to open a new ontology
KAON2Connection connection = KAON2Manager.newConnection();

// Mapping a virtual URI to its Physical URI
DefaultOntologyResolver resolver = new DefaultOntologyResolver();
resolver.registerReplacement(defaultNameSpace, inputURL.toString());
connection.setOntologyResolver(resolver);

// open ontology connection
ontology = connection.openOntology(defaultNameSpace, new HashMap());
```

Creating a new ontology ready for elements' additions

```
// new KAON2 connection
KAON2Connection connection = KAON2Manager.newConnection();

// setting up a ontology resolver that maps the logical
// uri to virtual uri
DefaultOntologyResolver resolver = new DefaultOntologyResolver();
resolver.registerReplacement(getSourceURI(), "file:" + outputFile.getName());
```

```
connection.setOntologyResolver(resolver);

// create an ontology by specifying its logical URI. The resolver
// provides the physical URI.
org.semanticweb.kaon2.api.Ontology ontology = connection.createOntology(
            getSourceURI(), new HashMap());
```

Methods from KAON2Utils

```
  /** rdfs:comment annotation */
  KAON2Utils.COMMENT;

  /** rdfs:label annotation */
  KAON2Utils.LABEL;

  /** rdfs:domain annotation */
  KAON2Utils.DOMAIN;

  /** rdfs:range annotation */
  KAON2Utils.RANGE;

  /**
   * Given an OWL Entity this method returns its name.
   * The value after '\#'
   * @param entity - entity whose name should be returned
   */
  KAON2Utils.getName(OWLEntity entity);

  /**
   * Each OWL entity can have one or more
   * annotations/attributes associated with
   * it. This method helps returning their values
   * by taking the following parameters.
   * @param annotation
   *          name of the annotation/attribute (
   *          e.g. rdfs:label, rdfs:comment etc)
   * @param oEntity
   *          OWLentity whose annotations need to be searched
   * @param ontology
   *          Ontology that contains the specified OWLEntity
   */
  KAON2Utils.getAnnotationValue(String annotation, OWLEntity oEntity,
          Ontology ontology);

  /**
   * Given an ontology it returns all available root
   * classes (classes with no other super class
   * @param ontology
   *          ontology from where to retrieve root classes
   * @return a set of OWLClass(es)
   */
  KAON2Utils.getRootClasses(Ontology ontology);

  /**
   * This method iterates through all Entities in
   * ontology and finds out only
   * those which are instances of OWLClass
   * @param ontology
   *          an instance of ontology
   * @return a set of all OWLClasses from the ontology
   */
```

```
KAON2Utils.getClasses(Ontology ontology);

/**
 * This method iterates through all Entities in
 * ontology and finds out only those which are
 * instances of either ObjectProperty or DataProperty
 * @param ontology
 *          an instance of ontology
 * @return a set of all Properties from the ontology
 */
KAON2Utils.getProperties(Ontology ontology);

/**
 * This method given an object property returns all
 * its super object
 * properties
 * @param op -
 *          object property
 * @param ontology -
 *          ontology that the object property belongs to
 * @param properties -
 *          where these properties should be added,
 *          if null return a new set
 * @param directClosure -
 *          if it should be in a direct closure or
 *          in a transitive closure
 */
KAON2Utils.getSuperObjectProperties(ObjectProperty op,
        Ontology ontology, Set properties, boolean directClosure);

/**
 * This method given a data property returns all
 * its super data properties
 * @param dp -
 *          data property
 * @param ontology -
 *          ontology that the data property belongs to
 * @param properties -
 *          where these properties should be
 *          added, if null return a new set
 * @param directClosure -
 *          if it should be in a direct closure or
 *          in a transitive closure
 */
KAON2Utils.getSuperDataProperties(DataProperty dp,
        Ontology ontology, Set properties, boolean directClosure);

/**
 * This method given an object property returns entity
 * classes that make up
 * its domain
 * @param op - object property
 * @param ontology - ontology where to search for
 *                   entity classes
 * @param directClosure
 */
KAON2Utils.getDomainDescriptions(ObjectProperty op,
        Ontology ontology, boolean directClosure);
/**
 * This method given a data property returns entity
 * classes that make up its
 * domain
 * @param dp - data property
 * @param ontology - ontology where to search for
 *                   domain
```

```
 * @param directClosure
 */
KAON2Utils.getDomainDescriptions(DataProperty dp,
        Ontology ontology, boolean directClosure);

/**
 * Given an individual instance, this method
 * returns all its properties (i.e.
 * object as well as data properties)
 */
KAON2Utils.getProperties(Individual individual,
                         Ontology ontology);

/** returns all object properties from the given ontology */
KAON2Utils.getObjectProperties(Ontology ontology);

/** returns all data properties from the given ontology */
KAON2Utils.getDataProperties(Ontology ontology);

/**
 * Given an OWLClass this method returns all its
 * super classes (direct closure)
 * @param oClass
 * @param ontology
 */
KAON2Utils.getSuperClasses(OWLClass oClass, Ontology ontology);

/**
 * Given an OWLClass this method returns all its subclasses
 * (direct closure)
 */
KAON2Utils.getSubClasses(OWLClass oClass, Ontology ontology);

/**
 * For a given OWLClass, this method returns all
 * individuals those belong to
 * this class
 * @param oClass
 * @param ontology
 */
KAON2Utils.getIndividuals(OWLClass oClass, Ontology ontology);

/**
 * This method returns a set of Classes that
 * this individual belongs to.
 * @param individual
 * @param ontology
 */
KAON2Utils.isInstanceOf(Individual individual, Ontology ontology);

/**
  * This method returns all subObject Properties of the
  * given Object property
  */
KAON2Utils.getSubObjectProperties(ObjectProperty objectProperty,
        Ontology ontology);

/**
  * This method returns all subDatatype properties of the given
  * Datatype property
  */
KAON2Utils.getSubDataProperties(DataProperty dataProperty,
        Ontology ontology);

/** Returns true if the given property is an ObjectProperty */
```

```
KAON2Utils.isObjectProperty(OWLEntity oEntity);

/** Returns true if the given property is aData Property */
KAON2Utils.isDataProperty(OWLEntity oEntity);

/**
 * Returns all instances from the ontology
 * @param ontology
 */
KAON2Utils.getIndividuals(Ontology ontology);

/**
 * Each OWL entity can have one or more
 * annotations/attributes associated with
 * it. This method helps adding new annotation by
 * taking the following parameters.
 * @param annotation -
 *          annotation name
 * @param value -
 *          annotation value
 * @param oEntity -
 *          entity for which the annotation to be included
 * @param ontology -
 *          ontology that the entity belongs to
 */
KAON2Utils.addAnnotation(String annotation, Object value,
        OWLEntity oEntity, Ontology ontology);

/**
 * This method given two classes adds subClassOfRelationship
 * @param subClass
 * @param superClass
 */
KAON2Utils.addSubClass(OWLClass subClass, OWLClass superClass,
        Ontology ontology);

/**
  * Adds a new individual member to the given class
  * @param individual
  * @param oClass
  */
KAON2Utils.addIndividual(Individual individual, OWLClass oClass,
        Ontology ontology);

/**
  * Add a new object property domain
  * @param op - object property
  * @param domainClass - instance of OWLClass
  * @param ontology - where to add it
  */
KAON2Utils.addObjectPropertyDomain(ObjectProperty op,
        OWLClass domainClass, Ontology ontology);

/**
  * Add a new object property range
  * @param op - object property
  * @param rangeClass - instance of OWLClass
  * @param ontology - where to add it
  */
KAON2Utils.addObjectPropertyRange(ObjectProperty op,
        OWLClass rangeClass, Ontology ontology);

/**
  * Adds a new data property domain
  * @param dp - data property
```

```
 * @param domainClass - instance of OWLClass
 * @param ontology - where to add
 */
KAON2Utils.addDataPropertyDomain(DataProperty dp,
        OWLClass domainClass, Ontology ontology);

/**
 * Add a data property member by specifing the member
 * individual and the value
 * for data property (e.g. cat has four legs, where cat
 * is an individual, having legs is a data property
 * with four as a value for data property)
 * @param dp
 * @param individual
 * @param dpValue
 */
KAON2Utils.addDataPropertyMember(DataProperty dp,
        Individual individual, Object dpValue, Ontology ontology);
/**
 * This method adds a new object property member
 * (e.g. john drinks water)
 * where
 * @param op
 *          drinks is an object property
 * @param subject
 *          john is a subject
 * @param object
 *          water is an object
 */
KAON2Utils.addObjectPropertyMember(ObjectProperty op,
        Individual subject, Individual object, Ontology ontology);

/**
 * Adding a subObject Property
 */
KAON2Utils.addSubObjectProperty(ObjectProperty subProperty,
        ObjectProperty superProperty, Ontology ontology);

/**
 * Creates a new owl class for the given class URI
 * @param owlClassURI
 */
KAON2Utils.createOWLClass(String owlClassURI);

/**
 * Creates a new object property for the given uri
 * @param objectPropertyURI
 */
KAON2Utils.createObjectProperty(String objectPropertyURI);

/**
 * Creates a new data property for the given uri
 * @param dataPropertyURI
 */
KAON2Utils.createDataProperty(String dataPropertyURI);

/**
 * Creates a new individual
 */
KAON2Utils.createIndividual(String individualURI);

/**
 * Given a uri, it returns its equivalent OWL class
 * @param uri
 */
```

```
KAON2Utils.getOWLClass(String uri);
```

Chapter 5

Report on Scheduled Tool Integration

In future versions of Text2Onto a graphical workflow engine will provide support for the automatic or semi-automatic composition of complex ontology learning workflows. For transforming the POM into a consistent OWL or RDF ontology we aim at a tight integration with the KAON evolution framework (initially created as part of [Sto04], extended in task 3.1 Incremental Ontology Evolution [HSV04b]) which will allow to detect and resolve inconsistencies in the generated POMs. The development of the explanation component will be carried on with particular regard to the DILIGENT methodology [PTS04]. By generating machine readable explanations we will make a major step in the direction of making Text2Onto part of the DILIGENT process. We are currently preparing an evaluation setting for comparing the results of the newly developed ontology learning algorithms with previous implementations provided by TextToOnto and other ontology learning tools. Moreover, a detailed user evaluation will offer valuable clues to the usability of the graphical user interface and the benefits gained from the availability of an explanation component.

Following the bottom-up integration of KAON2 within GATE reported here, in year 3 the tools will be used within the case studies, through components such as ontology-based information extraction (task 2.1). The outcomes from the user evaluation with the case studies will be used as a basis for identifying fruitful future integration points.

In other words, the year 2 bottom-up integration activities, reported here, were largely dictated by the needs of the technology workpackages, whereas we envisage that year 3 integration activities will largely focus on SIP-based component-level integration. For instance, integration between ontology-based IE, user profiling, and knowledge access. Therefore, further tightly-coupled, bottom-up integration will be carried out where the case studies provide a strong requirement for such activities.

Chapter 6

Conclusion

This deliverable is a part of a series of bottom-up deliverables D6.6.x on bilateral integration activities and provides information on the integration of the Text2Onto, GATE, and KAON2 tools.

Text2Onto is a framework for data-driven change discovery by incremental ontology learning. Text2Onto combines machine learning approaches with basic linguistic processing such as tokenization or lemmatization and shallow parsing. To perform linguistic processing it relies on GATE.

GATE is comprised of an architecture, a free open source framework (or SDK) and graphical development environment. It is used for all sorts of language processing tasks, including Information Extraction in many languages. As part of the GATE-Text2Onto integration, various GATE components such as tokenizer, stemmer, and part-of-speech tagger, have been integrated/used for linguistic processing in the Text2Onto system.

KAON2 is a complete infrastructure for managing OWL-DL and SWRL ontologies. It serves as the main ontology management infrastructure within SEKT and also as the backbone for ontology evolution functionalities within SEKT. At the same time, the GATE ontology support is aimed at providing an abstraction layer between the actual representation mechanism and the NLP modules making use of it.

The KAON2-GATE integration allows users to manage ontologies using KAON2 from GATE. This integration will enable the Ontology-Based IE tools (see deliverables D2.1.x) to access the SEKT ontologies via KAON2 and take this information into account when adding metadata to documents. In addition, deliverable D2.2.1 on Controlled Language Information Extraction has already benefited from this integration. The user is now able to create ontologies in natural language and save them within KAON2.

Bibliography

[BHS+05] Stephan Bloehdorn, Peter Haase, York Sure, Johanna Völker, Matjaz Bevk, Kalina Bontcheva, and Ian Roberts. Report on the integration of ml, hlt and om. SEKT Deliverable D6.6.1, Institute AIFB, University of Karlsruhe, JUL 2005.

[BNC00] G. Bisson, C. Nedellec, and L. Canamero. Designing clustering methods for ontology building - The Mo'K workbench. In *Proceedings of the ECAI Ontology Learning Workshop*, pages 13–19, 2000.

[BOS03] P. Buitelaar, D. Olejnik, and M. Sintek. OntoLT: A protégé plug-in for ontology extraction from text. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2003.

[CMBT02a] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

[CMBT02b] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Annual Meeting of the ACL*, 2002.

[CMT00] H. Cunningham, D. Maynard, and V. Tablan. JAPE: a Java Annotation Patterns Engine (Second Edition). Research Memorandum CS–00–10, Department of Computer Science, University of Sheffield, November 2000.

[Hea92] M.A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 539–545, 1992.

[HMS04a] U. Hustadt, B. Motik, and U. Sattler. Reasoning for Description Logics around $\mathcal{SHIQ}$ n a Resolution Framework. Technical Report 3-8-04/04, FZI, Karlsruhe, Germany, April 2004.
http://www.fzi.de/wim/publikationen.php?id=1172.

[HMS04b] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In D. Dubois, C. Welty, and M.-A. Williams, editors, *Proc. of the 9th Int. Conf. on Knowledge Representation and Reasoning (KR2004)*, pages 152–162, Menlo Park, California, USA, June 2004. AAAI Press.

[HSV04a] P. Haase, Y. Sure, and D. Vrandecic. Ontology mangement and evolution - survey, methods and prototypes. SEKT deliverable 3.1.1, Institute AIFB, University of Karlsruhe, 2004.

[HSV04b] P. Haase, Y. Sure, and D. Vrandecic. Ontology mangement and evolution - survey, methods and prototypes, sekt deliverable 3.1.1. Technical report, Institute AIFB, University of Karlsruhe, 2004.

[HV04] P. Haase and J. Voelker. Requirements analysis for usage-driven and data-driven change discovery. SEKT informal deliverable 3.3.1.a, Institute AIFB, University of Karlsruhe, 2004.

[Mae02] A. Maedche. *Ontology Learning for the Semantic Web*. Kluwer Academics, February 2002.

[May05] D. Maynard. Sekt deliverable d1.4.1, 2005. SEKT Deliverable.

[MS01] A. Maedche and S. Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2), 2001.

[MSS04] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proc. of the 3rd Int. Semantic Web Conf. (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 549–563, Hiroshima, Japan, 2004. Springer.

[PTS04] H. Sofia Pinto, Christoph Tempich, , and Steffen Staab. Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engingeering of ontologies. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 2004.

[Sto04] Ljlijana Stojanovic. Methods and tools for ontology evolution, phd thesis. Technical report, University of Karlsruhe, 2004.

[VNCN05] P. Velardi, R. Navigli, A. Cuchiarelli, and F. Neri. Evaluation of ontolearn, a methodology for automatic population of domain ontologies. In P. Buitelaar, P. Cimiano, and B. Magnini, editors, *Ontology Learning from Text: Methods, Applications and Evaluation*. IOS Press, 2005. to appear.

[VS05a] J. Voelker and Y. Sure. Data-driven change discovery. SEKT deliverable 3.3.1, Institute AIFB, University of Karlsruhe, 2005.

[VS05b] J. Voelker and Y. Sure. Data-driven change discovery. evaluation. SEKT deliverable 3.3.2, Institute AIFB, University of Karlsruhe, 2005.