D6.6.2 Integration of TextGarden with GATE

Matjaz Bevk (JSI), Kalina Bontcheva (USFD), Marko Grobelnik (JSI), Dunja Mladenic (JSI)

Abstract.
EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D6.6.2 (WP 6)

This deliverable is a part of the bottom up integration activities and focuses on bilateral integration of Text Mining and Human Language Technology. Deliverable gives overviews of TextGarden, a machine learning toolkit, and GATE, an infrastructure for Human Language Technology. Next, the use of machine learning tools for language processing is motivated and example scenarios are presented. Deliverable also gives reports on done and scheduled integration activities.

Keyword list: SEKT Integration, Text Mining, Human Language Technology

Document Id.	SEKT/2004/D6.6.2/v1.0
Project	SEKT EU-IST-2003-506826
Date	January 24, 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

Executive Summary

This deliverable first presents an overview of TextGarden, a Machine Learning toolkit (ML), and GATE, an infrastructure for Human Language Technology (HLT). Next, the use of machine learning tools for language processing is motivated and example scenarios are presented. Subsequently, the technical details of the bottom-up, tight coupling between TextGarden and GATE are presented, with a detailed example of use of SVM for information extraction.

The requirement for tight, i.e., bottom-up integration between HLT and ML comes from the high performance required from the combined system.

The integration of Human Language Technology tools with Machine Learning algorithms has a substantial benefit, as it enables the development of HLT systems based on Machine Learning, instead of hand-crafted rule-based ones. In general, the advantage of Machine Learning systems comes from the fact that they do not require expert users, i.e., a system based on learning can easily be retrained by a user who only needs to be an expert in the application domain and provide the necessary annotated training data. It also enables usage of ML systems on textual data taking into account language specific properties of the text provided by HLT either via pre-processing the data or using HLT as background knowledge in ML systems.

Last, but not least, this work has had impact on a number of related deliverables:

- D2.1.2 on Ontology-Based Information Extraction
- D2.5.2 on Evaluation of ontology-based IE
- D10.3.2 on Prototye for the SEKT Legal case study

Contents

Chapter 1

Introduction

SEKT aims at developing technologies for Next Generation Knowledge Management that exploit complementary research fields like Knowledge Discovery (KDD), 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 WP6 on, specifically task T6.6. It focuses on integration of SEKT technology based on KDD developed in WP1 and SEKT technology based on HLT developed in WP2. 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 over 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.

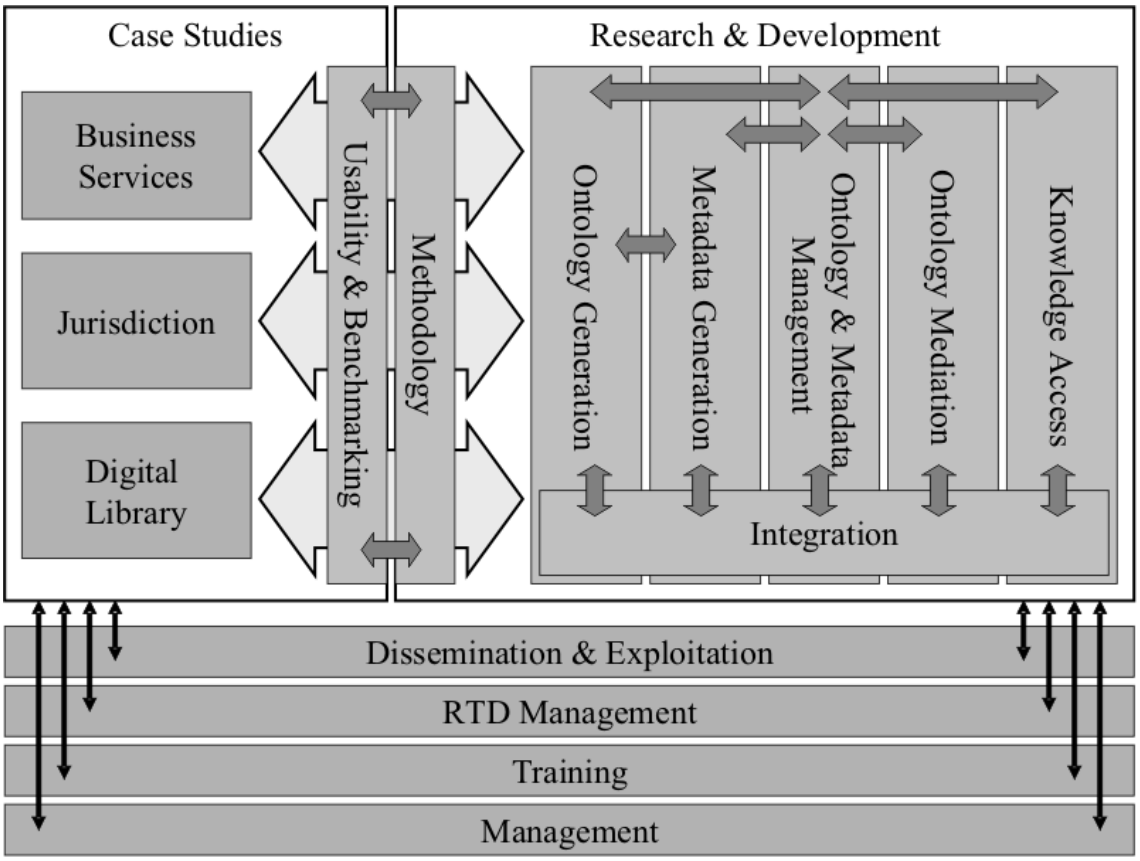

Figure 1.1: The SEKT Big Picture

1.2 Approaches to Integration in SEKT

Workpackage 6 aims at providing integration between the SEKT technical components in two ways. 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.

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 understand 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.

In general, the bottom-up integration activities are not meant to replace integration via SIP but aim at examining potential benefits from the combination of core technologies in a light-weight manner. We see the bottom-up integration as a great chance to gain

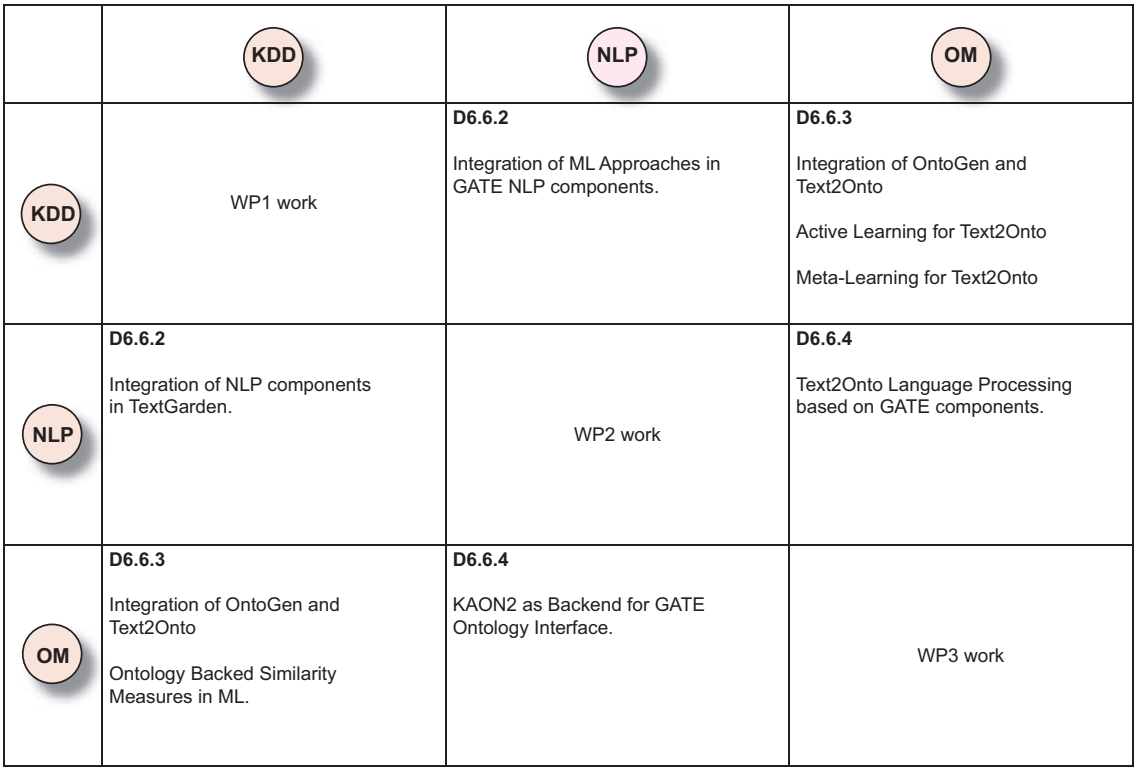

Figure 1.2: The SEKT Big Picture

insights on how to combine our different technologies. 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. Figure 1.2 gives an overview over the bilateral integration activities pursued within SEKT and the M24 deliverables within this task.

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 deliverable is structured as follows: chapter 2 gives overall descriptions of TextGarden and GATE, chapter 3 contains overview on integration scenarios, chapter 4 is a report on current tool integration, chapter 5 is a report on scheduled integration plans and finally chapter 6 concludes.

Chapter 2

Description of the Individual Components for Integration

To make this deliverable self-contained, we shortly describe the tools developed in other SEKT workpackages WP1 involving Knowledge Discovery and WP2 involving Human Language Technology. This chapter is meant as a reference for the next chapters that will refer back to these software tools.

2.1 Overview of TextGarden

Text Garden software tools for text mining is a set of software components, many of them developed mainly within SEKT (workpackage WP1 on Ontology Generation). The objective of this workpackage is to explore various aspects of generating ontological structures by means of machine learning, especially text mining methods. Text Garden tools enable easy handling of text documents for the purpose of data analysis including automatic model generation and document classification, document clustering, document visualization, dealing with Web documents, crawling the Web and many other. The code is written in C++ and originally runs on Windows platform. It is publicly availabel from `www.texmining.net`. TextGarden consists of a set of command line utilities, which could be run sequentially in pipeline manner to perform a specific learning task. Development of Text Garden started in 1996 [Mla96, GM98], with major revisions in late 90's. Several components for Text Mining were developed as a part of SEKT WP1 deliverables on the top of the existing library. Here we provide brief description of the software tools emphasizing the SEKT contributed parts where relevant.

Text Garden tools enable easy handling of text documents for the purpose of data analysis including: document pre-processing, feature construction, document classification and clustering, learning on unlabeled data and active learning, document visualzaition, focused crawling. The rest of this section describes briefly each of the functionalities that

are relevant for SEKT.

2.1.1 Document Pre-processing

Pre-processing of Text Documents in various formats. For SEKT the most relevant format is Bag-Of-Words format with the file extension `.Bow` This format corresponds to the commonly used representation of a text document with a word-vector ignoring position of words in the document.The purpose of the format is to enable efficient execution of algorithms working with the bag-of-words representation such as, clustering, learning, classification, visualization, etc.

2.1.2 Feature Construction

Feature construction components for learning semantic-space of documents that create semantic-space representation of documents based on Latent Semantic Indexing and enables projection of new documents on the learned semantic-space. And another component for feature construction from images (developed inside D1.3.1)

2.1.3 Document Classification

Document classification including automatic model generation based on various Machine Learning Algorithms including Support Vector Machines, k-Nearest Neighbor, Logistic Regression, Winnow and more. In SEKT deliverables so far we have used Support Vector Machines (SVM) and k-Nearest Neighbor. Document classification in Text Garden includes also document classification into a large topic ontology (developed inside D1.5.1) that is currently used in the BT case study on Inspec topic ontology.

2.1.4 Semi-supervised and Active Learning

Processing of unlabelled data and Active Learning (developed inside D1.2.1 [NGM04]). Active learning is implemented on sparse training sets using binary SVM model. It performs active learning loop on the specified input. Semi-Supervised transduction performs a transductive inference on a joint labelled and unlabelled dataset.

2.1.5 Document Visualization

Document Visualization [BFG05] Visualization of semantic-space of documents provides visualization of documents as a 2-D map based on the semantic-space representation (developed inside D1.4.1).

2.1.6 Focused Crawling

Focused crawling of the Web (developed inside D1.1.1 [NFMG04]) that exploits Google. Additionally there are separate components that enable getting one page from the web based on the Web address specified with URL.

2.2 Overview of GATE

GATE is implemented in Java and is available under the LGPL license. It is available from gate.ac.uk, including detailed user and programmer guides, Javadoc, and example code.

The GATE architecture distinguishes between data, algorithms, and their visualisation.[1] Following the terminology established in version 1, GATE components are one of three types:

- **Language Resources** (LRs) represent entities such as lexicons (e.g. WordNet), corpora or ontologies;[2]
- **Processing Resources** (PRs) represent entities that are primarily algorithmic, such as parsers, generators or n-gram modellers;
- **Visual Resources** (VRs) – added in version 2 – represent visualisation and editing components that participate in GUIs.

These resources can be local to the user's machine or remote (available over the (Inter)net), and all can be extended by users without modification to GATE itself.

One of the main advantages of separating the algorithms from the data they require is that the two can be developed independently by language engineers with different types of expertise, e.g. programming and linguistics. Similarly, separating data from its visualisation allows users to develop alternative visual resources, while still using a language resource provided by GATE.

Collectively, all resources are known as CREOLE (a Collection of REusable Objects for Language Engineering), and are described in XML configuration files, which declare their name, implementing class, parameters, icons, etc. This component metadata is used by the framework to discover and load available resources.

[1] Analoguous to the model/controller/view architecture common in GUI toolkits.

[2] Ontologies are often not considered as describing language and, therefore, their classification as LRs could be somewhat un-intuitive. Probably a more appropriate term will be data resources, but we adhere to the original terminology adopted in GATE v1. Fundamentally, GATE treats all declarative data resources like lexicons, ontologies, and documents in the same fashion, i.e., they can be provided as inputs or parameters to PRs and are visualised and edited in VRs.

Unlike version 1, GATE v2 offers comprehensive multilingual support using Unicode as its default text encoding. It also provides a means of entering text in various languages, using virtual keyboards where the language is not supported by the underlying operating platform [TUB+02].

When an application is developed within GATE's graphical environment, the user chooses which processing resources go into it (e.g. tokeniser, POS tagger), in what order they will be executed, and on which data (e.g. document or corpus). The application results can be viewed in the document viewer/editor. Applications can be saved, reloaded, and embedded in other systems.

All resource types have creation-time parameters that are used during the initialisation phase. Processing Resources also have run-time parameters that get used during execution.

Controllers are used to define GATE applications and have the role of controlling the execution flow.

2.2.1 GATE Machine Learning Framework

This section describes the use of Machine Learning (ML) algorithms in GATE for NLP tasks such as Information Extraction.

An ML algorithm "learns" about a phenomenon by looking at a set of occurrences of that phenomenon that are used as examples. Based on these, a model is built that can be used to predict characteristics of future (and unforeseen) examples of the phenomenon.

Classification is a particular example of machine learning in which the set of training examples is split into multiple subsets (classes) and the algorithm attempts to distribute the new examples into the existing classes.

This is the type of ML that is used in GATE and all further references to ML actually refer to classification.

2.2.2 Some definitions

- **instance**: an example of the studied phenomenon. An ML algorithm learns from a set of known instances, called a dataset.
- **attribute**: a characteristic of the instances. Each instance is defined by the values of its attributes. The set of possible attributes is well defined and is the same for all instances in a dataset.
- **class**: an attribute for which the values need to be found through the ML mechanism.

2.2.3 GATE-specific interpretation of the above definitions

- **instance**: an annotation. In order to use ML in GATE the users will need to choose the type of annotations used as instances. Token annotations are a good candidate for this, but any type of annotation could be used (e.g. things that were found by a previously run JAPE grammar).
- **attribute**: an attribute can be either:
  - the presence (or absence) of a particular annotation type [partially] covering the instance annotation
  - the value of a named feature of a particular annotation type.

  The value of the attribute can refer to the current instance or to an instance situated at a specified location relative to the current instance.
- **class**: any attribute can be marked as class attribute.

An ML implementation has two modes of functioning: training and application. The training phase consists of building a model (e.g. statistical model, a decision tree, a rule set, etc.) from a dataset of already classified instances. During application, the model built while training is used to classify new instances.

There are ML algorithms which permit the incremental building of the model (e.g. the Updateable Classifiers in the WEKA library). These classifiers do not require the entire training dataset to build a model; the model improves with each new training instance that the algorithm is provided with.

2.2.4 The Machine Learning PR in GATE

Access to ML implementations is provided in GATE by the ”Machine Learning PR” that handles both the training and application of ML model on GATE documents. This PR is a Language Analyser so it can be used in all default types of GATE controllers.

In order to allow for more flexibility, all the configuration parameters for the ML PR are set through an external XML file and not through the normal PR parameterisation. The root element of the file needs to be called ”ML-CONFIG” and it contains two elements: ”DATASET” and ”ENGINE”.

The DATASET element

The DATASET element defines the type of annotation to be used as instance and the set of attributes that characterise all the instances.

An ”INSTANCE-TYPE” element is used to select the annotation type to be used for instances, and the attributes are defined by a sequence of ”ATTRIBUTE” elements.

An ATTRIBUTE element has the following sub-elements:

- **NAME**: the name of the attribute
- **TYPE**: the annotation type used to extract the attribute.
- **FEATURE** (optional): if present, the value of the attribute will be the value of the named feature on the annotation of specified type.
- **POSITION**: the position of the annotation used to extract the feature relative to the current instance annotation.
- **VALUES**(optional): includes a list of VALUE elements.
- <**CLASS/**>: an empty element used to mark the class attribute. There can only be one attribute marked as class in a dataset definition.

Semantically, there are three types of attributes:

- **nominal attributes**: both type and features are defined and a list of allowed values is provided;
- **numeric**: both type and features are defined but no list of allowed values is provided; it is assumed that the feature can be converted to a number (a double value).
- **boolean**: no feature or list of values is provided; the attribute will take one of the ”true” or ”false” values based on the presence (or absence) of the specified annotation type at the required position.

Figure 2.1 gives some examples of what the values of specified attributes would be in a situation when ”Token” annotations are used as instances.

The ENGINE element

The ENGINE element defines which particular ML implementation will be used, and allows the setting of options for that particular implementation.

The ENGINE element has three sub-elements:

- **WRAPPER**: defines the class name for the ML implementation (or implementation wrapper). The specified class needs to extend gate.creole.ml.MLEngine.

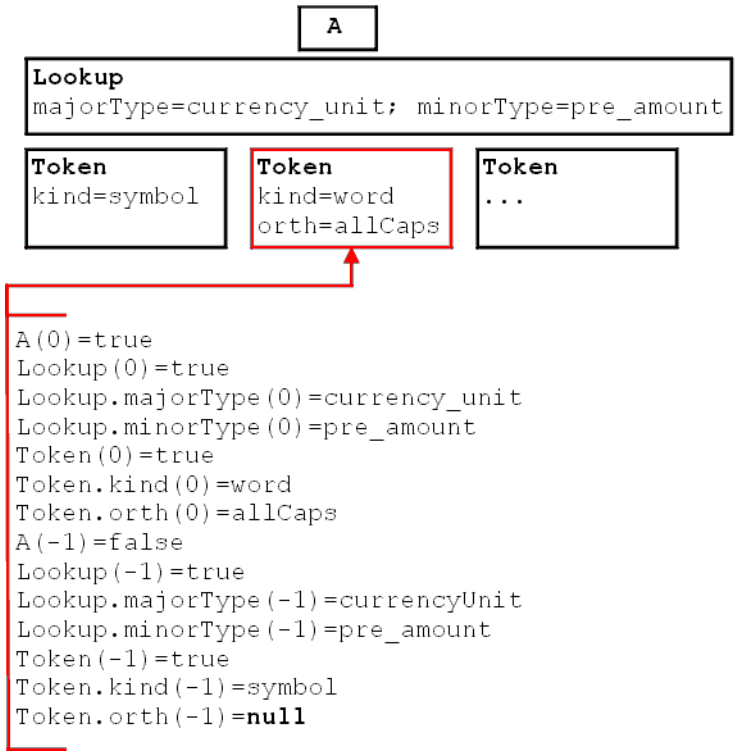

Figure 2.1: Sample attributes and their values

- **BATCH-MODE-CLASSIFICATION**: this element is optional. If present (as an empty element `<BATCH-MODE-CLASSIFICATION />`), the training instances will be passed to the engine in a single batch. If absent, the instances are passed to the engine one at a time. Not every engine supports this option, but for those that do, it can greatly improve performance.

- **OPTIONS**: the contents of the OPTIONS element will be passed verbatim to the ML engine used.

New ML engines, such as those provided by TextGarden, are integrated within Gate by implementing a new engine wrapper.

Chapter 3

Overview on Integration Scenarios

The rationale behind the integration of TextGarden's machine learning algorithms within GATE comes from the well-established need for adaptable and trainable NLP systems. One of the most widely used and successful ML algorithms for NLP tasks has been the SVM. Therefore, in the discussion below, we will focus on usage scenarios of this particular algorithm, although the points apply to any other ML approach, due to the generic nature of the ML framework in GATE.

Support Vector Machines (SVM) are a supervised machine learning algorithm, which has achieved state-of-the-art performance on many learning tasks. In particular, SVM is a popular learning algorithm for Natural Language Processing (NLP) tasks such as POS (Part-of-speech) tagging [JM03, NKM01], word sense disambiguation [LNC04], NP (noun phrase) chunking [KM00a], named entity recognition [IK02, MMP03], relation extraction [ZSZZ05], semantic role labeling [HPW$^+$04], and dependency analysis [KM00b, YM03]. Almost all these applications consist of three steps: first they transform the problem into one or more classification tasks; then an SVM classifier is trained for each task; and finally, the classifiers' results are combined to obtain the solution to the original NLP problem.

SVM is an optimal classifier in the sense that, given training data, it learns the maximal margin classifier which has good generalisation on unseen data. Consequently, on classification tasks SVMs tend to perform better than other distance- or similarity-based learning algorithms such as k-nearest neighbour (KNN) or decision trees. In addition, NLP tasks typically represent instances in very high dimensional feature vectors, which leads to positive and negative examples being distributed into two distinctly different areas of the feature space. This is particularly helpful for searching a classification hyperplane in feature space and for the generalisation capability of the classifier. Such very high dimensional representation is achieved by forming the feature vector explicitly from text and in many cases by exploring the so-called kernel function to map the feature vector into higher dimensional space (see e.g. [CST00]).

When compared to other classification problems, NLP classification tasks have sev-

eral unique characteristics, which are seldom considered in applications. Perhaps the most important one is that NLP tasks tend to have imbalanced training data, in which positive examples are vastly outnumbered by negative ones. This is particularly true for smaller data sets where often there are hundreds of negative training examples and only few positive ones. Another unique characteristic is that annotating text for training the algorithm is a time-consuming process, while at the same time unlabelled data is abundant. Therefore, when a learning algorithm is applied to NLP tasks, these particular aspects should be taken into account in order to obtain a practical system with good performance.

Therefore, in order to address these issues, TextGarden was integrated as a machine learning engine within GATE and SVMs are now widely used in Workpackage 2, for experiments on ontology-based information extraction (for further details see [LBC05a, LBC05b] and deliverables D2.1.1 and D2.1.2). Technical details of the integration effort appear in the following section.

Due to the processing power and efficiency required from the Information Extraction tools in SEKT, tight coupling between TextGarden and GATE was necessary. Interaction via SIP is more appropriate for bigger components with well-defined functionality, e.g., the IE tools, the knowledge access tool, the language generation tools, etc.

As already discussed above, the immediate beneficiary of the TextGarden-GATE integration is work in workpackage 2. The SEKT case studies will benefit indirectly from this, by using the ontology-based information extraction tools and being able to train and adapt them easily to new domains. For instance, there is ongoing work on the BT case study involvinh WP2 technology.

The rationale behind the integration of GATE's natural language processing into TextGarden comes from the fact that SEKT uses TextGarden components mainly to handle data written in natural language. In particular in the SEKT Legal case study, Spanish questions and judgements are processed. As Spanish has much more inflections compared to English, pre-processing the data using stemmer or lemmatizer is needed. That was also confirmed by our intial experiments on the legal case study data with no usage of natural language specific pre-processing.

TextGarden can be simply used via using command line utilities. Thus we have implemented integration between GATE and TextGarden via a java command line utility which exposes stemmer from GATE to the TextGarden.

As already pointed out, the immediate benefit from integrating human language technology tools into TextGarden is for the SEKT Legal case study, where the results of applying KDD technology are improved by natural language dependent data pre-processing.

Chapter 4

Report on Current Tool Integration

4.1 Integration of TextGarden tools into GATE

TextGarden consists of several tools, which can be used to perform various machine learning operations on a set of documents. A tool for running SVM algorithm `BowTrainBinSVM` has been integrated into GATE as well as the supporting tool for data pre-processing `Txt2Bow` and a tool for using a learned model `BowClassify`. A tool for running SVM assumes that the text was pre-processed as needed to get the right features and phrases. Actually, incorporating frequent phrases of n consequtive words is supported by Text2Bow utility (the maximum length of phrases is set via parameters).

- Txt2Bow:
  Transforms various raw text formats (in this case SVM light sparse format) into the file in Bag-Of-Words format `.Bow`
- BowTrainBinSVM:
  Learns a classification model via training a binary class Support Vector Machine on the set of input documents provided in the Bag-Of-Words format.
- BowClassify:
  Classifies a set of input documents in the Bag-Of-Words format using the provided model.

TextGarden tools are command line stand-alone executables implemented in C++. On the other hand GATE is a Java application. The technical part of integration of TextGarden into GATE is achieved by implementing Java wrappers which write out and read in the files expected by TextGarden tools and execute the TextGarden components. The TextGarden wrappers provide access to the TextGarden tools through the Machine Learning PR in GATE. The wrappers handle all temporary files internally and transparently for the GATE user, the user only interacts with GATE.

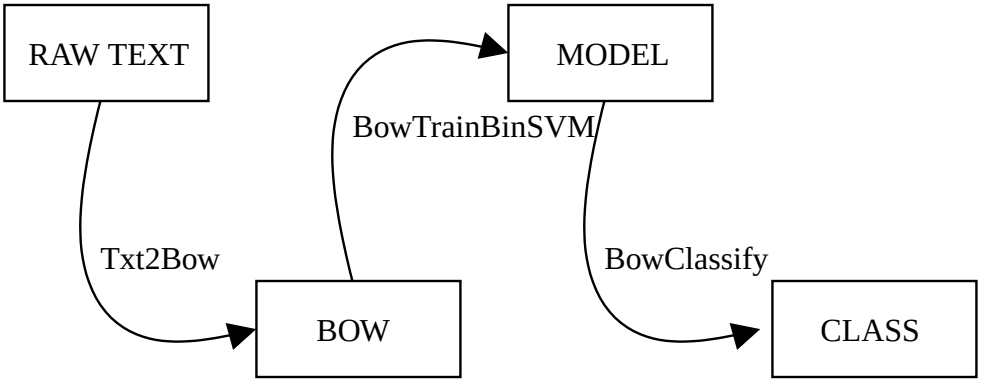

Figure 4.1: Usage diagram of the integrated TextGarden tools

The BowTrainBinSVM tool from TextGarden provides GATE with solutions for classification problems, as for example building a classification model on feature vectors for words. Here is a typical scenario in which TextGarden and GATE cooperate: GATE writes out to the file a description of a learning problem. Further on, GATE executes TextGarden utilities Txt2Bow and BowTrainBinSVM to obtain a model, which is later used to classify new instances using BowClassify utility from TextGarden.

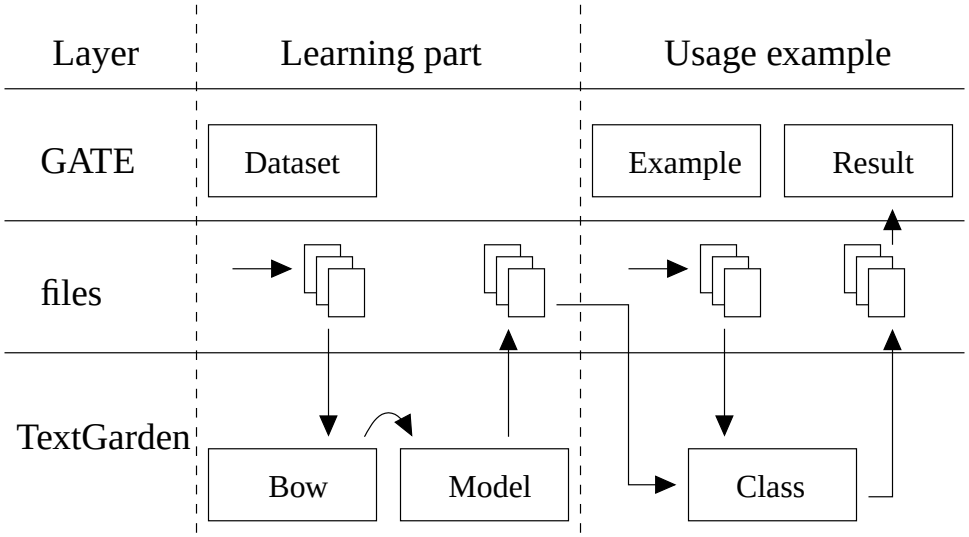

Figure 4.2: A typical scenario in which TextGarden and GATE cooperate.

The BowTrainBinSVM can only be used for binary classification, i.e. the target attribute (class) must be boolean. Of course all other attributes can be boolean, numeric or nominal or any combination of these. If an attribute is nominal each value of that attribute maps to a separate SVM feature. Each of these SVM features will be given the value 1 when the nominal attribute has the corresponding value, and will be set to 0 otherwise.

The TextGarden models are not updatable, and are so created and trained the first time a classification is attempted. However, the wrapper supports a batch classification mode of the machine learning processing resource. if <BATCH-MODE-CLASSIFICATION/> is specified in the <ENGINE/> part of the configuration file, then all instances for a document will be passed to the wrapper at one time, rather than them being passed one at a time. Using this option greatly improves efficiency in most circumstances.

Here are the options for the wrapper:

- TRAINER-OPTIONS: These are options that are passed to the BowTrainBinSVM tool on the command line.

  The tool learns binary Support Vector Machine (SVM) classifier on the input file `-i` for classifying documents into one category `-cat`. It produces model `-o` in Bag-Of-Words format `.BowMd`. Both positive and negative examples are needed for learning. Input vectors can be weighted `-w` with different weights.

  The parameter `-c` determines the value of cost parameter for SVM, which must be greater than 0. Cost parameter can be weighted differently for positive and negative examples with parameter `-j` ($C^+ = jC, C^- = C$). The parameter `-t` selects kernel used for learning:

  1. 0 - linear kernel (much faster than others)
  2. 1 - polynomial kernel $k(x, y) = (s(x^T y) + c)^p$
  3. 2 - radial kernel $k(x, y) = e^{(-gamma\|x-y\|^2)}$
  4. 3 - sigmoid kernel $k(x, y) = tanh(sx^T y + c)$

  Parameters `-ker_p`, `-ker_s`, `-ker_c` and `-ker_gamma` determine parameters of nonlinear kernels.

  The parameter `-cachesize` determines size of cache (in MB) non-linear SVM can use for caching evaluated kernel functions. The parameter `-time` determines maximal time in seconds allowed for learning classifier. If the optimal hyperplane is not found in given time, then most optimal hyperplane found so far is returned as a result. The parameter `-v` determines verbosity during learning. The parameters `-subsize` determines size of sub-problems used at learning algorithm (-1 means classifier decides). The parameters `-ter` determines termination criteria. By increasing it learning gets faster but at the end classifier is less accurate. The parameter `-shrink` determines if support vectors are prediction while learning. Using this option can increases learning time dramatically

- CLASSIFIER-OPTIONS: There are options that are passed to the BowClassify tool on the command line.

  The BowClassify tool uses a provided model `-imd` to classify a new instance `-qh` and outputs the result of the classification to the xml file `-ox`.

4.1.1 Example SVM Use in GATE

Classifying documents using the SVMWrapper is a two phase procedure. In its first phase, SVMWrapper collects data from the pre-annotated documents and builds the SVM model using the collected data to classify the unseen documents in its second phase. Below we describe briefly an example of classifying the start time of the seminar in a corpus of email announcing seminars and provide more details later in the section.

Figure 4.3 explains step by step the process of collecting training data for the SVM classifier. GATE documents, which are pre-annotated with the annotations of type *Class* and feature *type='stime'*, are used as the training data.

In order to build the SVM model, we require start and end annotations for each *stime* annotation. We use pre-processor JAPE transduction script to transform each *stime* annotation into two annotations - one marking the start, called *sTimeStart*, and one marking the end, called *sTimeEnd*. For instance, if the text contains an annotation spanning the string '2 p.m.', then it is replaced by two annotations: one *sTimeStart* which covers the string '2' and one *sTimeEnd*, which covers the '.' at the end. Then two SVM classifiers are trained – one for the start and one for the end. Further details on this process can be found in deliverable D2.1.1.

Following this step, the Machine Learning PR (SVMWrapper) with training mode set to true collects the training data from all training documents. GATE corpus pipeline, given a set of documents and PRs to execute on them, executes all PRs one by one only on one document at a time. Unless provided in a separate pipleline, it makes it impossible to send all training data (i.e. collected from all documents) altogether to the SVMWrapper using the same pipeline to build the SVM model. This results into the model not being built at the time of collecting training data. The state of the SVMWrapper can be saved to an external file once the training data is collected.

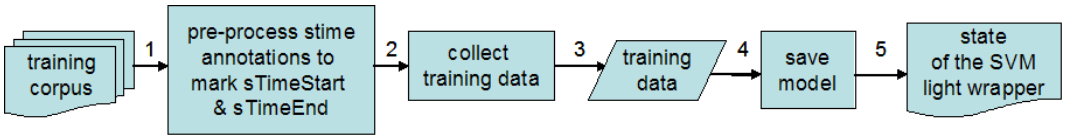

Figure 4.3: Flow diagram explaining the SVM training data collection

Before classifying any unseen document, SVM requires the SVM model to be available. In the absence of an up-to-date SVM model, SVMWrapper builds a new one using a command line *SVM_learn* utility and the training data collected from the training corpus. In other words, the first SVM model is built when user tries to classify the first document. At this point the user has an option to save the model somewhere on the external storage. This is in order to reload the model prior to classifying other documents and to avoid rebuilding of the SVM model everytime the user classifies a new set of documents. Once the model becomes available, SVMWrapper classifies the unseen documents which creates new *sTimeStart* and *sTimeEnd* annotations over the text. Finally, a post-processor

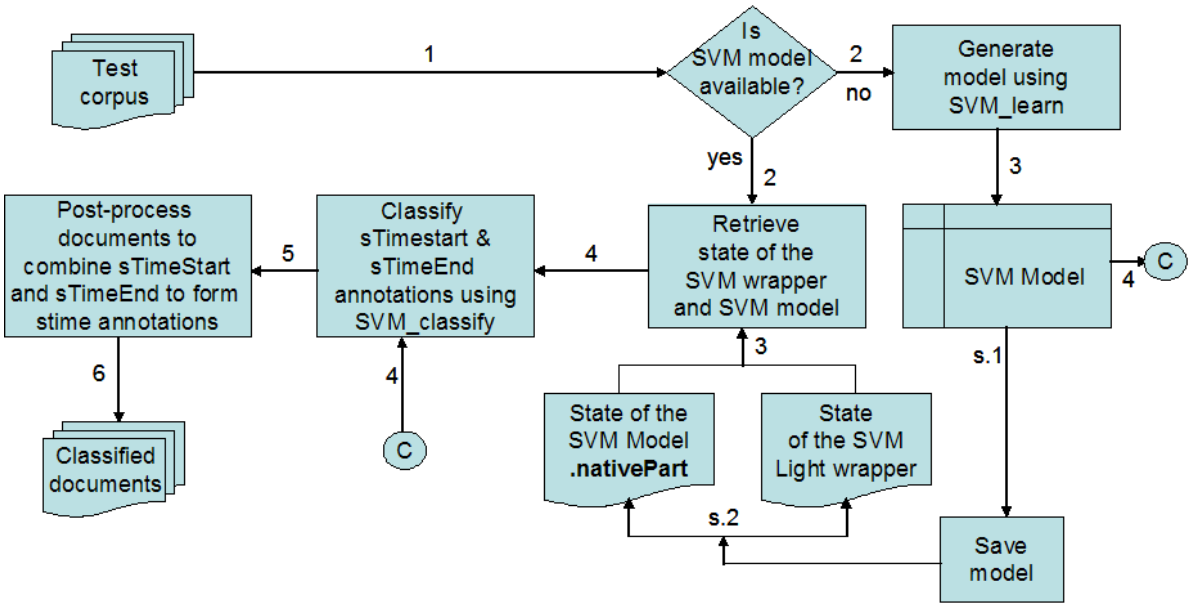

Figure 4.4: Flow diagram explaining document classifying process

JAPE transduction script is used to combine them into the *sTime* annotation. Figure 4.4 explains this process.

The wrapper allows support vector machines to be created which either do boolean classification or regression (estimation of numeric parameters), and so the class attribute can be boolean or numeric. The class attribute can be a three value nominal, in which case the first value specified for that nominal in the configuration file will be interpreted as *true*, the second as *false* and the third as *unknown*.

The other attributes can be boolean, numeric or nominal, or any combination of these. If an attribute is nominal, each value of that attribute maps to a separate SVM feature. Each of these SVM features will be given the value 1 when the nominal attribute has the corresponding value, and will be omitted otherwise. If the value of the nominal is not specified in the configuration file or there is no value for an instance, then no feature will be added. Text Garden models are not updateable, and so are created and trained the first time a classification is attempted. However, the Wrapper supports the batch classification mode of the machine learning processing resource. If <BATCH-MODE-CLASSIFICATION/> is specified in the <ENGINE> part of the configuration file, then all the instances for a document will be passed to the wrapper at one time, rather than them being passed one at a time. Using this option will result in a great improvement in efficiency in most circumstances.

4.2 Integration of GATE functionalities with TextGarden

Stemmer, which is a CREOLE plugin for GATE, has been integrated with TextGarden. Since TextGarden consists of several command line utilities, we followed this principle and we implemented a glue `TG2GStem` between GATE and TextGarden as a java command line utility which exposes stemmer from GATE to the TextGarden. The short-term plan is to use this utility to easily integrate Spanish lemmatiser from GATE with TextGarden once lemmatiser is available for GATE.

4.2.1 Using TG2GStem

TG2GStem is a java application. It is executed from command line. It takes at least two arguments on input. The first argument is a language specification for stemmer. <language> can be one of the following options:

- english
- finnish
- french
- german
- italian
- norwegian
- portuguese
- russian
- spanish
- swedish

The second option is an input <document> to be processed. There may be more then one document specified on the command line.

Once executed `TG2GStem` initializes GATE and executes two CREOLE plugins sequentially Tokenizer and Stemmer. The result is written to a <document>.stem.xml file.

4.2.2 Setting up TG2GStem

For smooth failsafe operation of `TG2GStem` GATE should be properly installed, CLASSPATH variable should include at least `gate.jar` archive and libraries from GATE's `lib` directory. Additionally a Java system property `gate.home` should point to the GATE installation directory.

Here is and example of executing `TG2GStem`:

```
SET GATE_HOME=C:\Program Files\GATE 3.0
java -cp ".;%GATE_HOME%\bin\gate.jar;
    %GATE_HOME%\lib\gnu-regexp-1.0.8.jar;
    %GATE_HOME%\lib\jdom.jar;
    %GATE_HOME%\lib\xerces.jar;
    %GATE_HOME%\lib\ontotext.jar;
    %GATE_HOME%\lib\jasper-compiler-jdt.jar"
    -Dgate.home="%GATE_HOME%"
    TG2GStem english process_me.txt
```

In this example the output would be written to `process_me.txt.stem.xml`.

Chapter 5

Report on Scheduled Tool Integration

Following the bottom-up integration activities 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 only be carried out if the case studies provide a strong requirement for such activities.

Chapter 6

Conclusion

This deliverable first presented an overview of TextGarden, a machine learning toolkit, and GATE, an infrastructure for Human Language Technology. The use of machine learning tools for language processing was motivated and example scenarios were presented. Subsequently, the technical details of the bottom-up, tight coupling between TextGarden and GATE were presented, with a detailed example of use of SVM for information extraction.

This work has had impact on a number of related M24 deliverables:

- D2.1.2 on Ontology-Based Information Extraction
- D2.5.2 on Evaluation of ontology-based IE
- D10.3.2 on Prototye for the SEKT Legal case study

Bibliography

[BFG05] Dunja Mladenic Blaz Fortuna and Marko Grobelnik. Visualization of text document corpus. *Informatica journal*, 29(4):497–502, 2005.

[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.

[CST00] N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[GM98] Marko Grobelnik and Dunja Mladenić. Learning machine : design and implementation. Jsi technical report (ijs-dp-7824), Jožef Stefan Institute, 1998.

[HPW+04] K. Hacioglu, S. Pradhan, W. Ward, J. H. Martin, and D. Jurafsky. Semantic Role Labeling by Tagging Syntactic Chunks. In *Proceedings of CoNLL-2004*, pages 110–113, 2004.

[IK02] H. Isozaki and H. Kazawa. Efficient Support Vector Classifiers for Named Entity Recognition. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02)*, pages 390–396, Taipei, Taiwan, 2002.

[JM03] J.Gimenez and L. Marquez. Fast and Accurate Part-of-Speech Tagging: The SVM Approach Revisited. In *Proceedings of the International Conference RANLP-2003 (Recent Advances in Natural Language Processing)*, pages 158–165. John Benjamins Publishers, 2003.

[KM00a] T. Kudo and Y. Matsumoto. Use of Support Vector Learning for Chunk Identification. In *Proceedings of Sixth Conference on Computational Natural Language Learning (CoNLL-2000)*, 2000.

[KM00b] T. Kudoh and Y. Matsumoto. Japanese Dependency Structure Analysis Based on Support Vector Machines. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 2000.

[LBC05a] Y. Li, K. Bontcheva, and H. Cunningham. SVM Based Learning System For Information Extraction. In *Proceedings of Sheffield Machine Learning Workshop*, Lecture Notes in Computer Science. Springer Verlag, 2005.

[LBC05b] Y. Li, K. Bontcheva, and H. Cunningham. Using Uneven Margins SVM and Perceptron for Information Extraction. In *Proceedings of Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, 2005.

[LNC04] Y. Lee, H. Ng, and T. Chia. Supervised Word Sense Disambiguation with Support Vector Machines and Multiple Knowledge Sources. In *Proceedings of SENSEVAL-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 137–140, 2004.

[Mla96] Dunja Mladenić. Personal webwatcher: Implementation and design. Jsi technical report ijs-dp-7472, Jožef Stefan Institute, 1996.

[MMP03] J. Mayfield, P. McNamee, and C. Piatko. Named Entity Recognition Using Hundreds of Thousands of Features. In *Proceedings of CoNLL-2003*, pages 184–187. Edmonton, Canada, 2003.

[NFMG04] Blaž Novak, Blaž Fortuna, Dunja Mladenić, and Marko Grobelnik. Collecting data for ontology generation. SEKT deliverable 1.1.1, Jožef Stefan Institute, 2004.

[NGM04] Blaž Novak, Marko Grobelnik, and Dunja Mladenić. Dealing with unlabelled data. SEKT deliverable 1.2.1, Jožef Stefan Institute, 2004.

[NKM01] T. Nakagawa, T. Kudoh, and Y. Matsumoto. Unknown Word Guessing and Part-of-Speech Tagging Using Support Vector Machines. In *Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium*, 2001.

[TUB+02] V. Tablan, C. Ursu, K. Bontcheva, H. Cunningham, D. Maynard, O. Hamza, Tony McEnery, Paul Baker, and Mark Leisher. A Unicode-based Environment for Creation and Use of Language Resources. In *3rd Language Resources and Evaluation Conference*, 2002. `http://gate.ac.uk/sale/iesl03/iesl03.pdf`.

[YM03] H. Yamada and Y. Matsumoto. Statistical Dependency Analysis with Support Vector machines. In *The 8th International Workshop of Parsing Technologies (IWPT2003)*, 2003.

[ZSZZ05] G. Zhou, J. Su, J. Zhang, and M. Zhang. Exploring Various Knowledge in Relation Extraction. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 427–434, 2005.