



D2.2.1 Report: Controlled Language IE Components version 1

**Tamara Polajnar, Hamish Cunningham, Valentin Tablan,
Kalina Bontcheva
(University of Sheffield)**

Abstract.

EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D2.2.1 (WP2.2)

This deliverable outlines an approach to automatic generation of metadata from natural language. The emphasis is on easy to use applications and controlled language interfaces.

In brief, Controlled Language Information Extraction (CLIE) aims to use natural language processing technology and simple mark-up to enable users to create, administer and use knowledge repositories stored in repositories like KAON and SESAME. For the application testing an authoring environment is devised using the wiki technology invented by Ward Cunningham, and Yet Another Mark-up (YAM), a mark-up language derived from Terrence's Mark-up Language (TML).

Keyword list: controlled language, information extraction, language processing

Document Id.	SEKT/2005/D2.2.1/v1.0
Project	SEKT EU-IST-2003-506826
Date	January 17, 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

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 AI EAD, 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 Autònoma 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

A controlled language is a subset of a natural language which is generally designed to be less ambiguous than the complete language and to include only certain vocabulary terms and grammar rules which are relevant for a specific task. Controlled languages have a long history of use. Since the 1970s CLs have been used by large companies such as Caterpillar, Boeing, and Perkins for generation of multilingual documentation.

Although most machine translation systems use some sort of intermediate representation for the information contained in the sentences which are being translated, the use of CLs for knowledge management is a relatively new field with first applications appearing in the mid 1990s. [Sow02] shows that all languages have the power to express first order logic statements. The problem usually is that natural language is far more powerful than first order logic. Beyond introducing shades of doubt or a range of logical values which can be represented with fuzzy sets, it has the power to express emotions and allusions to culture which are difficult if not impossible for a computer to comprehend. Constraining the language to avoid grammatically, contextually, and logically ambiguous statements leads to a great improvement in parsing and production of conceptual data. Though controlled languages can restrict the colourfulness of expression, they can be used to efficiently communicate concrete information. In most cases using a CL is an exercise in expressing information more consistently and concisely.

Controlled Language Information Extraction (CLIE) aims to use CL technology and simple mark-up to enable users to create, administer and use knowledge repositories stored in repositories like KAON and SESAME. For the application testing an authoring environment is devised using the wiki technology invented by Ward Cunningham, and Yet Another Mark-up (YAM), a mark-up language derived from Terrence's Mark-up Language (TML).

Contents

1	Introduction	3
1.1	Controlled Languages	3
2	Overview	5
2.1	Controlled Languages (CLs)	5
2.2	Experimental Apparatus	6
3	YAM	8
3.1	Introduction	8
3.2	Features	8
4	The Wiki	15
4.1	Introduction	15
4.2	GATE Wiki	15
4.2.1	Wiki design	16
4.3	On-line vs. off-line	17
5	CLIE	19
5.1	Introduction	19
5.2	Controlled Languages	20
5.2.1	CLs for Knowledge Engineering	20
5.2.2	Other CLs	21
5.3	CLIE language	22
5.3.1	Description of the CL	22
5.3.2	How to use CLIE CL	25
5.3.3	How to use CLIE through GATE	27
5.3.4	Design	29
5.4	Summary and Future Work	30
A	Yam Syntax	32
A.1	Overview of JAPE	32
A.2	Yam Syntax	34
A.2.1	Bold, italic and teletype	34
A.2.2	Horizontal lines	34

A.2.3	Lists	35
A.2.4	Verbatim output	35
A.2.5	Notes	36
A.2.6	Escapes	36
A.2.7	Headings	36
A.2.8	Links and anchors	36
A.2.9	Quotations	37
A.2.10	Line breaks	37
A.2.11	Tables	38
A.3	YAM syntax design	38
A.3.1	Bugs:	38
A.3.2	Wish list:	39
A.3.3	WikiLinks	40
A.4	CLIE implementation	40
A.4.1	clie.jape	40
A.4.2	np.jape	60
A.4.3	quotes.jape	63

Chapter 1

Introduction

People organise, compartmentalise, and order their environment. The world of computing is rife with structured information. Object-oriented programs, databases and file-systems are common examples of structured data. Controlled Language Information Extraction (CLIE) is an application which will allow users to design, create, and manage information spaces without knowledge of complicated standards such as XML or RDF. The CLIE architecture is divided into two independent parts: the language interface and the application.

The language interface builds on research from machine translation and elsewhere in use of controlled natural languages for knowledge representation. The application interface is based on common lightweight technologies which are familiar to web users. With the growing interest in Semantic Web applications and need to port information into a machine readable format there are many uses for this type of application.

1.1 Controlled Languages

Creating formal data is a high initial barrier to entry for small organisations and individuals wishing to create ontologies and thus benefit from semantic knowledge technologies. Part of the answer is in ontology authoring tools such as Protege, which however tend to require specialist knowledge about ontology engineering. Therefore, in the case of naive users, the definition of a controlled language for formal data description will enable them to author ontologies directly in natural language. Building on controlled language MT work, IE for controlled language analysis may achieve the high levels of accuracy necessary to make this viable.

The controlled language IE task has developed a simplified natural language processor that, on the one hand, allows the specification of logical data for Semantic KT purposes in normal language, while, on the other hand, attaining the high accuracy levels necessary for high-reliability applications. The components are based on GATE's existing FST cascade

IE [CMBT02]. CLIE is configured so that it either accepts input as valid (in which case accuracy is in general 100%) or rejects it and warns the user of needed repairs to their syntax. In certain cases the system will try a less strict analysis mode in order to suggest how repair may be effected. Therefore, because the parsing process is deterministic, accuracy is not really an issue.

Chapter 2

Overview

With the amount of information each person receives each day in various formats it is becoming increasingly difficult to manage, store, and retrieve personal data. It is possible to use the controlled language technologies to create an application which would allow users to use the most intuitive interface, natural language, to process, tag, and sort their data without the intimate knowledge of complex applications or knowledge representation languages or tools.

2.1 Controlled Languages (CLs)

A controlled language is a subset of a natural language which is generally designed to be less ambiguous than the complete language and to include only certain vocabulary terms and grammar rules which are relevant for a specific task.

Early controlled languages can trace their roots to 1970's Caterpillar Fundamental English (CFE). Caterpillar Inc. is a worldwide producer of complex mechanics. As a result it has a need to create documentation for the international market. The first approach to universal documentation representation lead to the creation of CFE, a subset of English with a limited vocabulary of 850 terms. CFE was easier to understand around world; however, for various practical reasons Caterpillar decided to produce multilingual documentation. This gave rise to Caterpillar Technical English (CTE). CTE is much more expressive than CFE. It has over 70,000 carefully chosen domain-specific terms. CFE grammar limits ambiguity, and thus allows easier machine translation into multiple languages. More recently, Caterpillar has begun merging with the KANTOO system developed at Carnegie Mellon University in order to improve grammar analysis and system performance [KAMN98].

Since the advent of CFE there have been several derivatives including Smart's Plain English Program (PEP) and White's International Language for Serving and Maintenance (ILSAM). Many of the world's largest companies including IBM, Rank Xerox, Boeing,

Perkins Engines and Ericsson Telecommunications are involved in CL research [AS92].

An interest has also been expressed in using CLs in knowledge technologies. Computer Processable English (CPE), developed at Cambridge Computer Laboratory, is an attempt to devise a controlled language which could be easily translated into a knowledge representation language. The project has been successful in easing the process of creating domain specific knowledge bases [Pul96]. ClearTalk is a language used by a series of knowledge engineering tools created at Ottawa University. The tools can extract knowledge from ClearTalk automatically. In addition, tools to assist the production of ClearTalk from unrestricted language also exist [Sku03], [Sow02]. The KANT system also has the ability to produce conceptual graphs and OWL statements from its intermediate representation [KAMN98]. [Sow02] also demonstrates that there is a mapping between the database query language SQL and controlled languages.

Controlled languages are further discussed in Section 5.2.

CLIE CL, which is introduced in Section 5.3.1, is similar to the above controlled languages in that it is limited. However, it is different than most of the above mentioned languages in that it does not limit vocabulary and the rules in the same way. For example, CTE strives to limit the term ambiguity in the lexicon, as well as sentence ambiguity in the parser. CLIE CL instead allows a very small number of sentences which contain particular key-phrases. This significantly reduces the number of grammatical constructions and sentences a user has to learn.

CLIE CL is created for a very specific task of ontology generation. To ensure maximum accuracy only a few non-ambiguous sentence types are considered. Languages used for generation, like CTE, need more expressibility as the text produced using these languages is for human consumption. CLIE CL is meant for human production, and for that reason each sentence has an easily computable outcome in the ontology. The vocabulary in CLIE CL is unlimited when it comes to assigning names to classes, properties, and other ontological elements; however, a user cannot use verbs which are outside of the permitted range. CLIE CL differs from languages like CTE, ClearTalk, and KANT, but is similar to languages which correspond to a limited symbolic language like first order logic.

2.2 Experimental Apparatus

To experiment on CLIE we defined a system which combines information organisation with CLs to create an application which is simple and accessible.

The application interface aims to be as user friendly as possible, while remaining lightweight. We adapt wiki technologies for this purpose. Wikis are web-based, and thus portable, user interfaces which have text based interfaces which require little or no training to use.

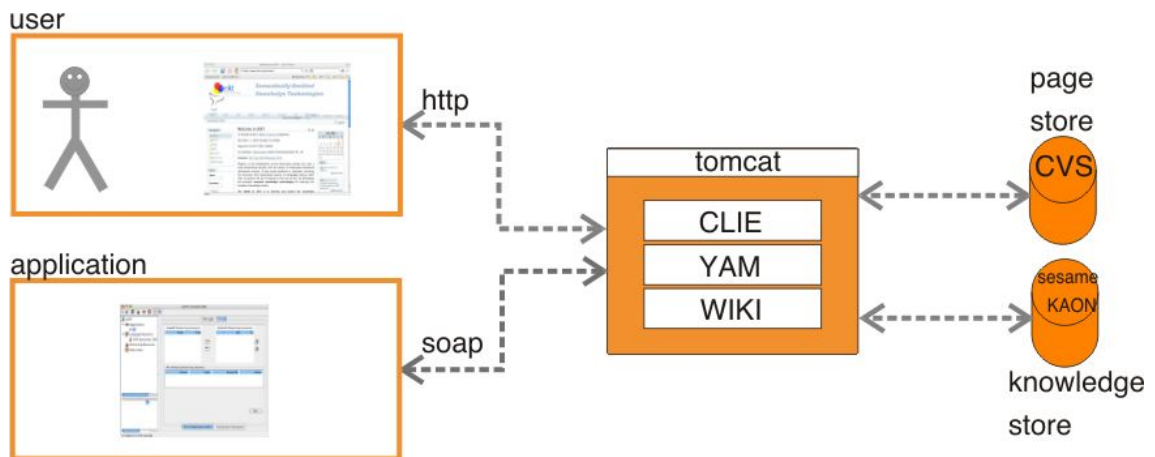


Figure 2.1: The CLIE test environment

The environment is modular and the CL interface can be accessed by other applications. Each of the parts: the Wiki, YAM, and CLIE are independent.

Chapter 3

YAM

3.1 Introduction

Yet Another Mark-up (YAM) is derived from Terrance’s Mark-up Language¹. It embodies the simplicity of wiki editing languages, but it is open to extension through the use of plug-ins. See Table 3.1 for an example of YAM code. Similarly as TML it is implemented using Antlr. Antlr² (ANother Tool for Language Recognition) is a tool supporting the creation of parsers, compilers, and generators for languages such as Java, C++, and YAM (in our case).

The YAM translator is easily extended with new output languages. Currently it can produce output in HTML and LaTeX and thence PDF. The UML description of the YAM module (overall) is shown in Figure 3.1. The figures following Figure 3.1 are parts of it, shown clearly, for better understanding. They are shown in order as Part 1 (Figure 3.2-Top Left), Part 2 (Figure 3.3-Bottom Left), Part 3 (Figure 3.4-Top Right) and Part 4 (Figure 3.5-Bottom Right)

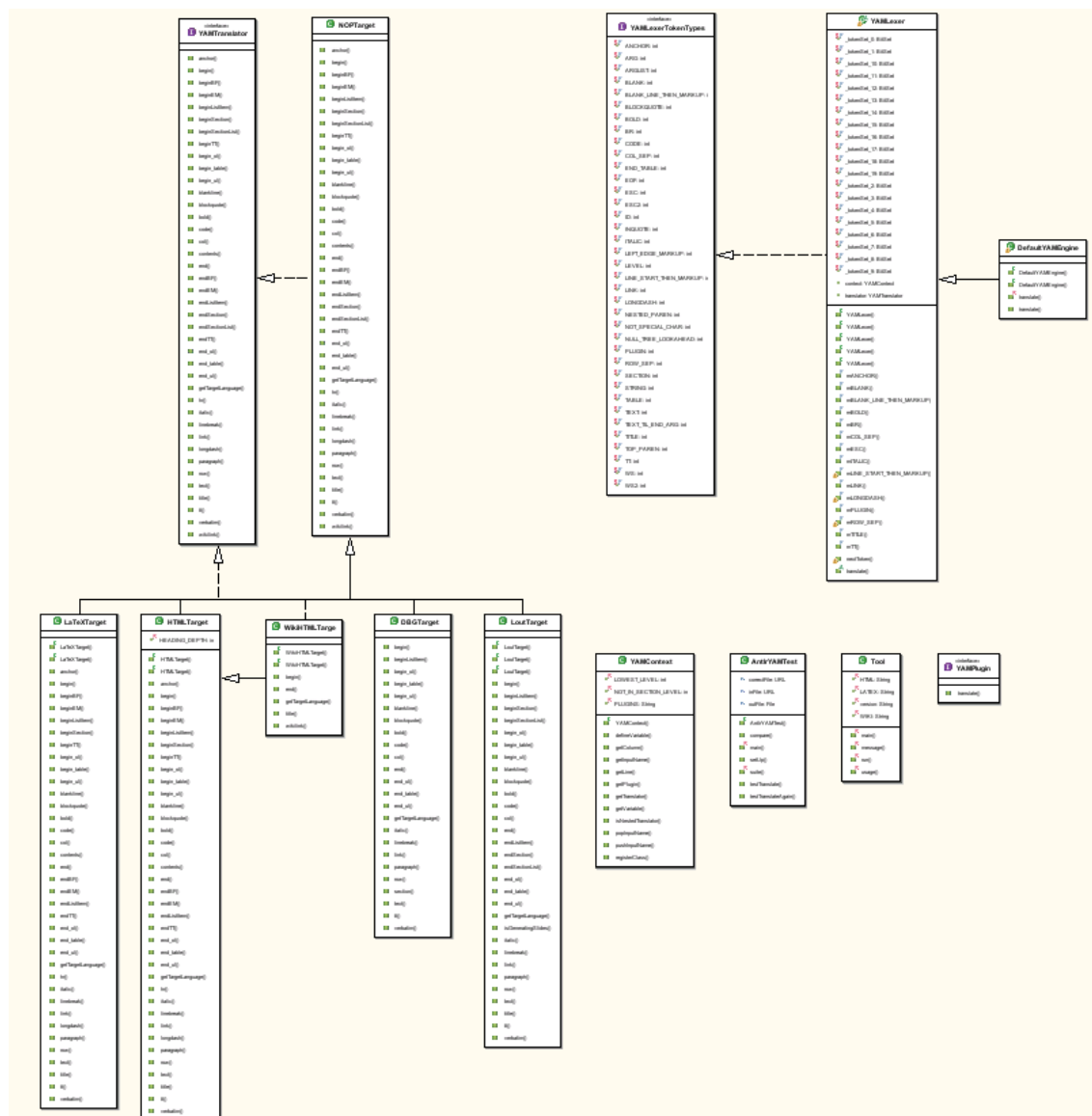
3.2 Features

Simple YAM is rich in features. From simple text processing features like **bold**, *emphasis*, and `teletype` to tables, a title page, and a table of contents, there is a range of features which can help create highly presentable documents in both LaTeX and HTML:

- Headings
- Text processing: **bold**, *emphasis*, `teletype`

¹<http://wwwantlr.org/TML/index.tml>

²<http://wwwantlr.org/>



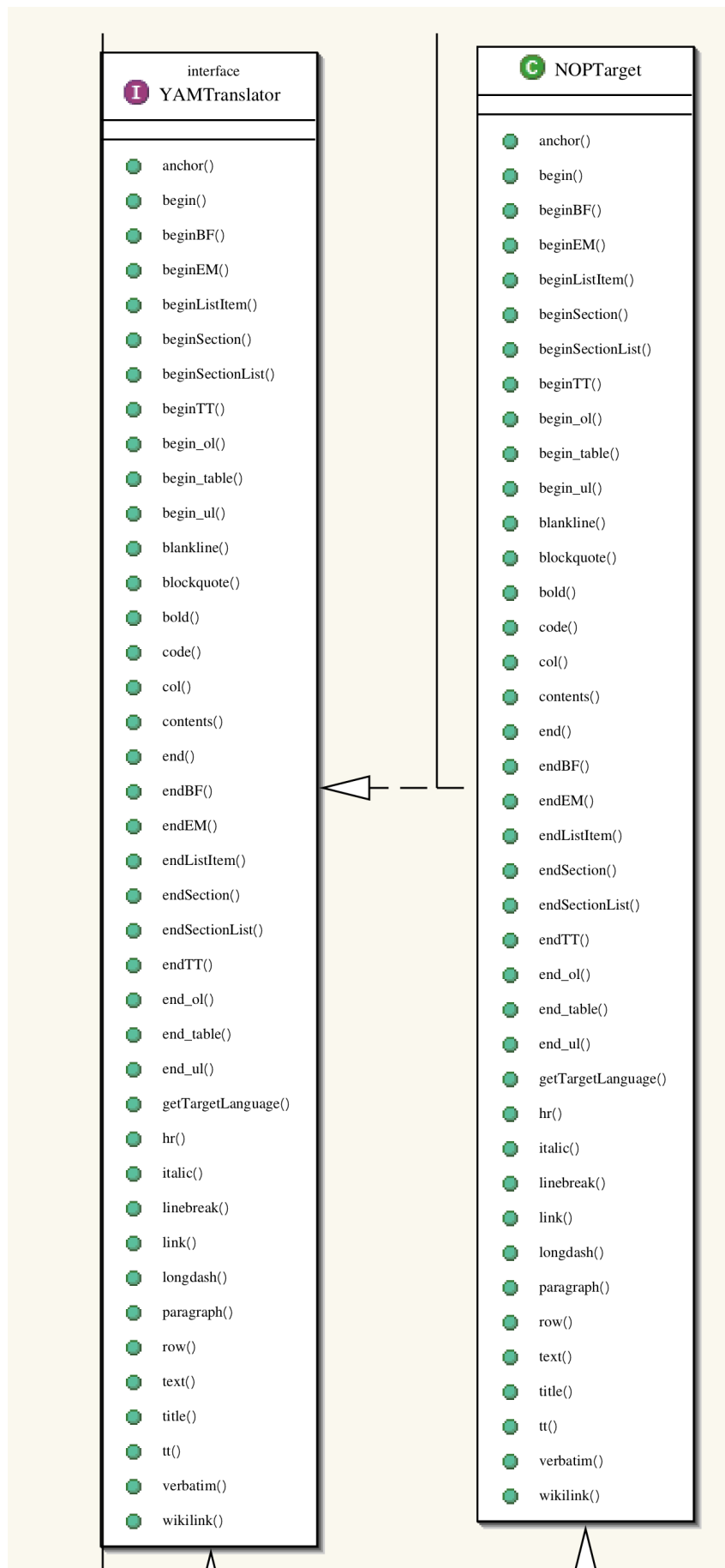


Figure 3.2: Part 1

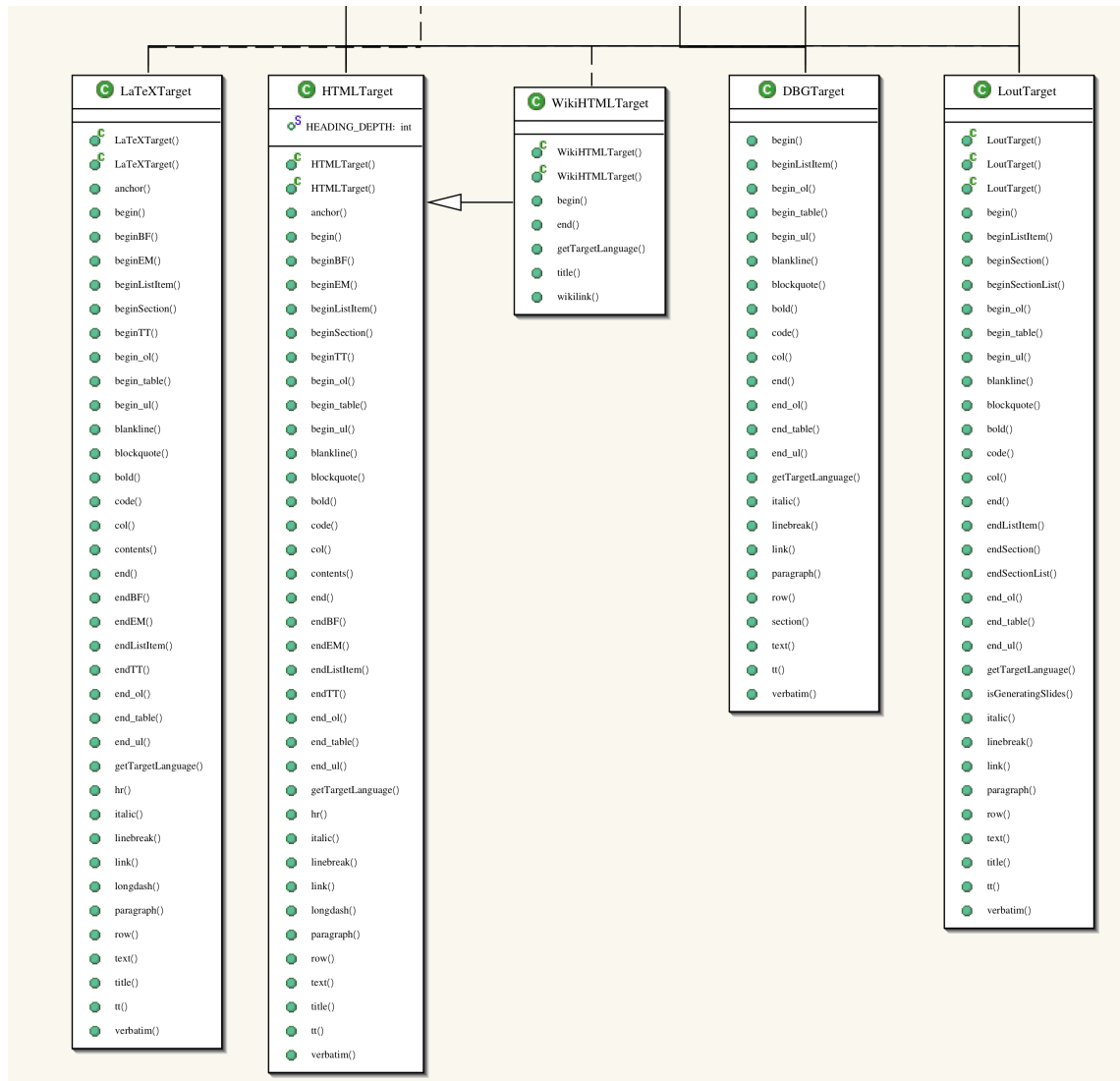


Figure 3.3: Part 2

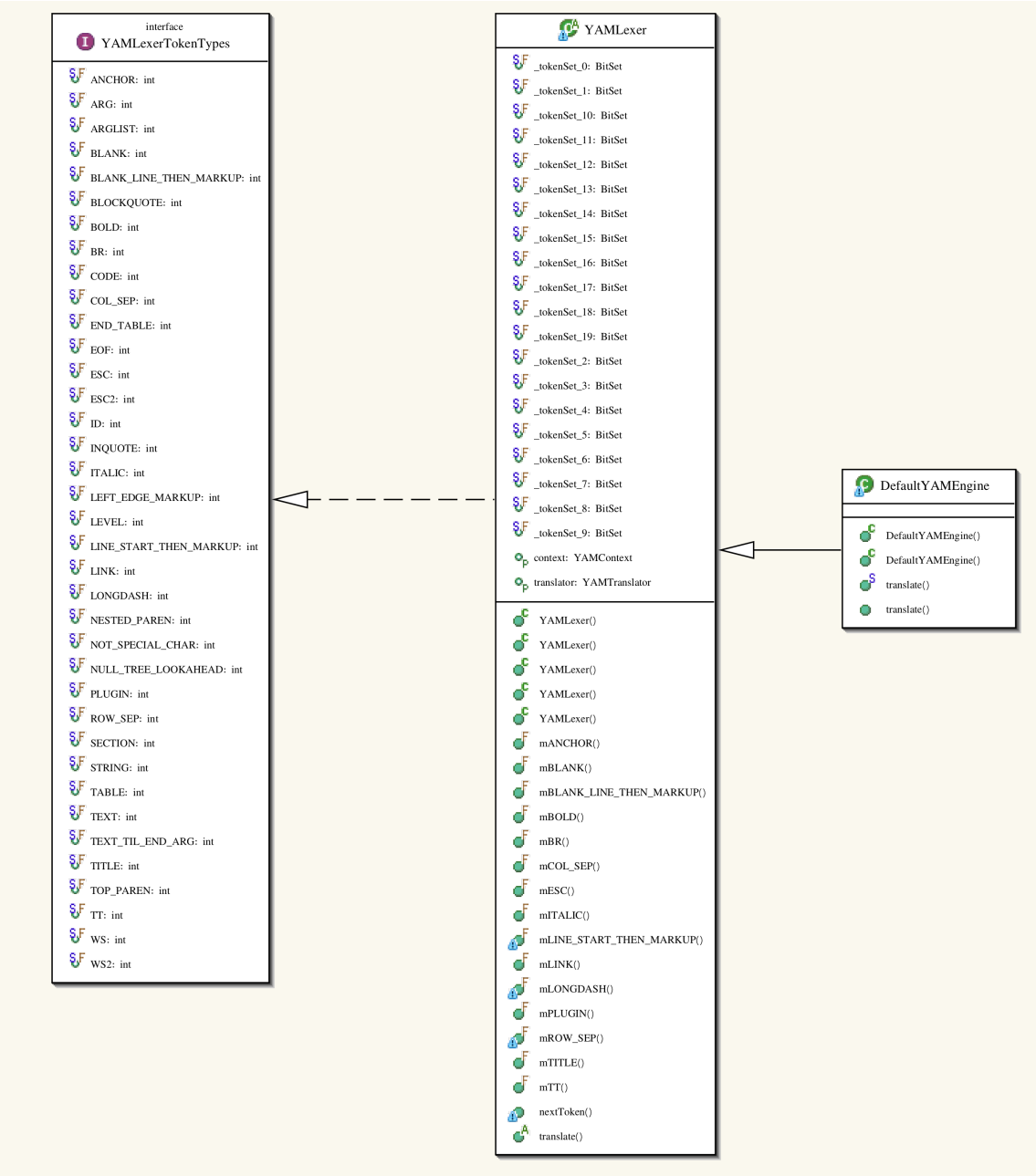


Figure 3.4: Part 3

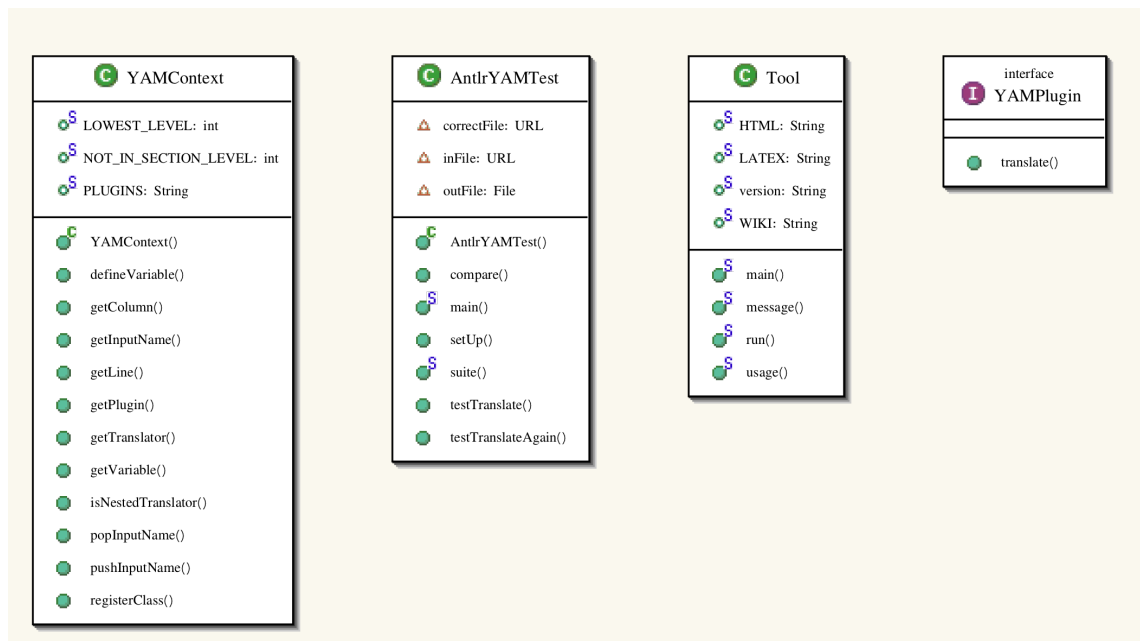


Figure 3.5: Part 4

- Line breaks
- Hrules
- Nested Lists
- Links and targets
- Tables
- Verbatim
- Quotations
- Plugins

Plug-ins in particular are very powerful and allow different extensions such as comments, boxed text, date and time support, etc.

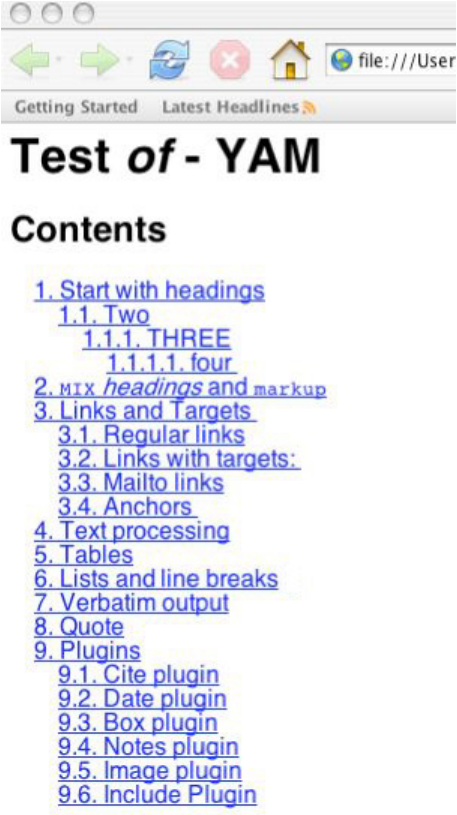
<div>Code:</div> <div>Test _of_ - YAM</div> <div>%contents</div> <div>%1 Start with headings</div> <div>%2 Two</div> <div>%3 THREE</div> <div>%4 four</div> <div>Generates:</div> <div></div> <div>1. Start with headings</div> <div>1.1. Two</div> <div>1.1.1. THREE</div> <div>1.1.1.1. four</div>	<div>Code:</div> <div>%1 Lists and line breaks</div> <div>- list item one</div> <div>\</div> <div>plus new para within bullet.\</div> <div>newline in para\</div> <div>graph.</div> <div>- list item two</div> <div>that wraps and stuff</div> <div>o nested item one</div> <div>o nested</div> <div>item two</div> <div>- back</div> <div>Generates :</div> <div>6. Lists and line breaks</div> <div>• list item one</div> <div>plus new para within bullet.</div> <div>newline in para</div> <div>graph.</div> <div>• list item two that wraps and stuff</div> <div>1. nested item one</div> <div>2. nested item two</div> <div>• back</div>
--	--

Table 3.1: Example of YAM

Chapter 4

The Wiki

4.1 Introduction

The concept of the WikiWikiWeb was invented by Ward Cunningham in 1995¹. The name *wiki-wiki*, which originally comes from Hawaiian, is a synonym for *quick* and has been popularly shortened to wiki. The idea behind the wiki technology is perhaps best embodied in Ward's famous quote: "The simplest online database that could possibly work." Wikis are characterised by simple editing languages which are automatically translated into HTML, thus speeding up page production. It is probably the quickest way to create a web-site which is text centred. Wikis generally allow all users to freely edit pages which has lead to very successful free-content systems and web-sites like Wikipedia². Wikipedia is one of the largest repositories of knowledge on the internet³. Anyone with internet access can add new entries and amend information already posted. This has lead to access to information not ordinarily available in traditional encyclopaedias, such as biographies of popular icons. Wikipedia also actively appears in over 100 languages.

4.2 GATE Wiki

The simple nature of the wiki makes it perfect as a quick testing ground for CLIE. The interface it provides is lightweight and portable. It is also very easy to use, so it will not get in the way of testing or evaluation of the overall system.

The wiki is implemented in Java and JSP which also adds to portability. It stores its pages in a CVS or SVN repository which allows version support and synchronisation for off-line editing. The wiki itself does not depend on the input language and they function

¹<http://c2.com/cgi/wiki?WikiHistory>

²http://en.wikipedia.org/wiki/Main_Page

³<http://en.wikipedia.org/wiki/Wikipedia>

separately. The chosen input language is YAM.

Why another Wiki, instead of reusing an existing one? Various reasons are stated below:

- The available implementations in languages that we know around here are either large and complex, or undocumented, or both. Therefore the effort involved in their reuse is liable to be as great or greater than reimplementation of a simple system.
- No available Wiki has good CVS support. Using CVS as a backend gives us:
 - Off-line edit - simply checkout the pages and edit to your heart's content while off-line.
 - Edit with other tools, not just the (horrible) web forms interface.
 - Management of authorship-related metadata such as how many lines added, difference between versions and so on.
 - A stable and reliable versioning system that's been proved in production use by 000,000s of developers.
 - Concurrent editing.
- Wikis are pretty simple, and their design well understood, so it is not too much effort to reimplement.
- The majority of SEKT components are implemented in Java, especially the GATE ones, so development in Java would be most suitable from integration perspective.
- JSPWiki comes close, but the input language is parsed in an ad-hoc manner which makes it difficult to alter.

4.2.1 Wiki design

To design the wiki server we need to think about the lifecycle of the following model components (the wiki as a whole has a Model-View-Component (MVC) architecture⁴):

- PageSets
- WikiPages
- WikiLinks
- Repositories

⁴For further details on MVC architectures see <http://ootips.org/mvc-pattern.html>

PageSets are implemented as file trees, i.e. top-level directories in the server's wiki filesystems. The location of these filesystems is a configuration option to the server (i.e. the server maintains a list of directories to treat as wiki filesystems, where every top-level directory represents a PageSet). PageSets are stored in a Repository; the copy that the wiki server maintains is checked out from the Repository.

PageSets are the top-level entry points to the wiki (like TWiki's "webs") that will be listed by the server's top-level page.

WikiPages are stored as `.yam` files that live in a PageSet. (Other files in the directory tree are simply served as normal.) A request for a `.html` or `.pdf` that doesn't exist but shares the same name as the WikiPage returns a translation of the `.yam`. During translation controlled language sections will trigger callouts to the CLIE interpreter.

WikiLinks are relative links within WikiPages that reference WikiPages. They differ from normal links in that the target doesn't necessarily exist. When a non-existent link target is found during the translation of a `.yam`, a link to a "create me" page is inserted. This means that the translator has to be given a list of all WikiPages in the current PageSet.

PageSets can be manipulated off-line by checking them out from the Repository and then using a normal editor and the YAM compiler application. Changes are fed back to the wiki server by way of the Repository when back on-line. Also, a wiki server can be run locally when off-line.

The Repository backends use a copy/edit/merge protocol (i.e. there is no locking during edit). This means that editors must resolve conflicts manually, and that the wiki server must keep PageSets up-to-date. This can be done in either of two ways:

1. Repository roots are configured to notify the wiki server of checkin events. Such events relating to `.yam` files trigger (re-)generation of the `.html/.pdf` outputs; for other files they trigger updating of the checked-out copy.
2. A background process in the server periodically checks the repository for updates and does translations as necessary.

Repositories are initialised relative to root descriptors, which are just normal CVS/SVN root identifiers and may therefore be local or remote.

4.3 On-line vs. off-line

One of the advantages of the design is that most page requests (for HTML and PDF) can be satisfied by the normal static page mechanism. The generated pages are kept up-to-date when the server is on-line in response to callbacks from the version control subsystem. When the server is off-line pages will go out of date; two things have to happen as a response:

- when going on-line, the server should update and regenerate any changed pages;
- the server overrides the error page from the web server; when a request comes for a translation that hasn't been done, the server checks for a `.yam` and does the translation, bringing it up-to-date

Chapter 5

CLIE

5.1 Introduction

CLIE stands for Controlled Language Information Extraction. CLIE aids in creation and maintenance of metadata. Instead of extracting metadata from existing sources CLIE would make it easier to create and maintain ontologies and other structured information spaces.

Two of the possible uses for CLIE include creation of a structured repository (and its maintenance) and populating an existing repository created by another application. CLIE interacts with knowledge repositories such as Sesame or KAON. The input into CLIE can be plain text or mark-up language such as YAM. The advantage of using something like YAM is that a user can better indicate the structure of the information using lists and tables.

Motivations for a CLIE OWL authoring facility:

1. The difficulty of writing OWL:
Sure and Volz report that of the OWL ontologies on the web in early 2004 the large majority were in OWL Full (the most complex of the OWL variants, which has rather high complexity and correspondingly poor tool support). However, it turned out on further examination that around 70% of these were actually OWL DL or OWL Lite ontologies which contained mistakes that made them into OWL Full. They also report that 90% of OWL ontologies are relational or taxonomic.
2. The usual knowledge engineering bottlenecks:
Pulman 1996 cites acquisition, transparency and maintainability as areas in which controlled language can help.
3. Others are now looking at programmable Wikis, e.g.
JotSpot from Excite's founders:

“Simple Web applications are not simple to build,” Kraus said in a statement. “We’ve taken the advantages of traditional document-based wikis — designing as you go and the wiki conforming to a user’s unique work style rather than the other way around... “We make wikis programmable, allowing you to layer structured information on top of all the unstructured data.”

5.2 Controlled Languages

Other than for machine translation, CLs are used for enforcement of standards for technical documents. For example ASD Simplified English (formerly AECMA Simplified English) is used by aerospace maintenance documentation, but is also used by other industries¹. SMART Controlled English is used worldwide as a simplified form of English which can be learned easier than full English². SMART enforces lexical disambiguation which makes it easily adaptable for machine translation. Many of the widely accepted CL standards come with tools for authoring, vocabulary generation, spell checking, etc.

Although most CL-based machine translation systems use some sort of intermediate representation for the information contained in the sentences which are being translated, the use of CLs strictly for knowledge management is a relatively new field with first applications appearing in the mid 1990s. [Sow02] shows that all languages have the power to express first order logic statements. The problem usually is that natural language is far more powerful than first order logic. Beyond introducing shades of doubt or a range of logical values which can be represented with fuzzy sets, it has the power to express emotions and allusions to culture which are difficult if not impossible for a computer to comprehend. Constraining the language to avoid grammatically, contextually, and logically ambiguous statements leads to a great improvement in parsing and production of conceptual graphs. Though controlled languages can restrict the colourfulness of expression, they can be used to efficiently communicate concrete information. In most cases using a CL is an exercise in expressing information more consistently and concisely.

5.2.1 CLs for Knowledge Engineering

Three of the systems which are to various degrees able to represent knowledge are KANT, ClearTalk, and Attempto Controlled English.

The Knowledge based Accurate Natural language Translation system or KANT is a system developed at CMU and is used for machine translation and question answering. The system uses a set of lexicons, grammars and syntactic rules in order to perform machine translation. The main application for KANT was for translation of highly technical documents. The newer versions of KANT have capabilities to convert controlled

¹<http://www.boeing.com/phantom/sechecker/se.html>

²<http://www.smartny.com/controlledEnglish.htm>

language representations to concept graphs and OWL statements. They also provide an interactive controlled language checker to help the user reformulate the sentences that do not conform to the KANT rules [KAMN98].

ClearTalk³ is a constrained language which can be converted to any KR notation easily. ClearTalk is an expressive CL that looks quite close to natural English. The author describes a system in which 25000 different facts from various domains are built into a ClearText Knowledgebase. The CT documents themselves can be in the form of simple web pages and any search engine can be used to index these pages. Once the CT knowledgebase is built, rather than using just simple keywords, we can query the system for more semantic concepts. There is a set of about one hundred rules that need to be understood in order to formulate CT documents [Sku03].

Attempto Controlled English (ACE) is designed for requirements specification and knowledge representation. Attempto is a research project from the University of Zurich. It is a restricted grammar written as a DCG in Prolog. The vocabulary consists of predefined determiners and other function words, where the nouns, adjectives, verbs, and other open word classes are domain specific, and specified by a user. Attempto is directly translatable into first order logic [Sch00].

Other research projects looking into CLs and KR include ⁴:

- Common Logic Controlled English (CLCE)⁵ aims to come as close to English as possible while being translatable to FOL.
- First Order English (FOE)⁶ is an English representation of first order logic.
- Controlled English to Logic Translation (CELT is a small CL which takes present indicative verbs and singular nouns into KIF ontologies) [PM03].

5.2.2 Other CLs

Other implementations of CLs which are used for document production and machine translation (MT): Perkins Approved English (PACE), SMART, ASD Simplified English (formerly AECMA Simplified English), IBM Easy English, Suns Controlled English, and Avayas Controlled English. The advantage of using CLs for MT is that pre-editing of the text in order to remove any lexical, syntactic or semantic ambiguities improves the accuracy and speed of machine translation.

[AS92] have developed a CL grammar for NLP research called COGRAM that consists of about 150 controlled English rules. A computer implementation of this grammar is

³<http://www.factguru.com>

⁴<http://www.ics.mq.edu.au/rolfs/controlled-natural-languages/>

⁵<http://www.jfsowa.com/clce/specs.htm>

⁶http://www.clp.ox.ac.uk/people/staff/pulman/current_projects.html

ALCOGRAM. ALCOGRAM was implemented for research purposes as a way to investigate NLP technologies, in particular grammar checking and computer aided language learning. COGRAM consists of a basic word list containing 5000 words and an additional 1000 technical terms and a simple set of grammatical rules. There are three main components of COGRAM, the lexical component, the syntactic component and the stylistic component. The ALCOGRAM version combines this component with algorithmic grammars that are used for verifying and formulating the text in a controlled language.

Given the ubiquity of controlled languages it is interesting to note that most of the languages were developed in a different way. The approaches vary in size of the dictionary, number of grammatical rules, and extensibility. [AS92] emphasises the lack of similarity between ASD (AECMA), Ericsson and IBM grammars. The important factor for this may be that CLs usually tend to be domain dependent and are also designed keeping the application in mind and can hence be quite subjective.

Below is a summary of the various features of some CLs.

CL	CFE	SMART	KANT	COGRAM	CT
Dictionary size	800	1200	?	5000	?
Total Rules	?	?	-	150	100
Technical documents	Yes	Yes	Yes	Yes	Yes
Machine Translation	Some	?	Yes	Yes	?
KR support	No	?	OWL	?	OWL/RDF/CG
Validator	?	Yes	Yes	Yes	?

The CLIE language is implemented using GATE [CMBT02].

5.3 CLIE language

Our CL is a simple way to construct and maintain knowledge. It allows a user to type in plain English sentences a description of a knowledge base and its contents.

5.3.1 Description of the CL

CLIE CL is modelled to allow maximum expressibility within the smallest set of syntactic structures. The limited number of allowed syntactic sentence structures makes the language easier to learn; CLIE is much easier to use than OWL, RDF, or SQL.

CLIE employs a deterministic approach to translation, so that if a sentence can be parsed, it can be parsed in one way only. Allowed sentences are unambiguous, so each sentence type is translated to one type of statement. If parsing fails it will result in a warning and no ontological output. A user can easily verify which sentences

have been translated and which have not by using GATE annotations. The annotation `CLIE-SentenceParsed` marks all sentences which were successfully translated. In this way, if a sentence conforms to the rules of the language it results in a predictable and accurate output. In some cases if the structure of the sentence conforms to one of the defined structures, but the ontological inputs such as class names or properties are not recognised as valid noun phrases, the grammar will attempt less rigorous parsing. The user will be warned if this is the case.

For example the text:

There are projects. There are workpackages, tasks, and deliverables.

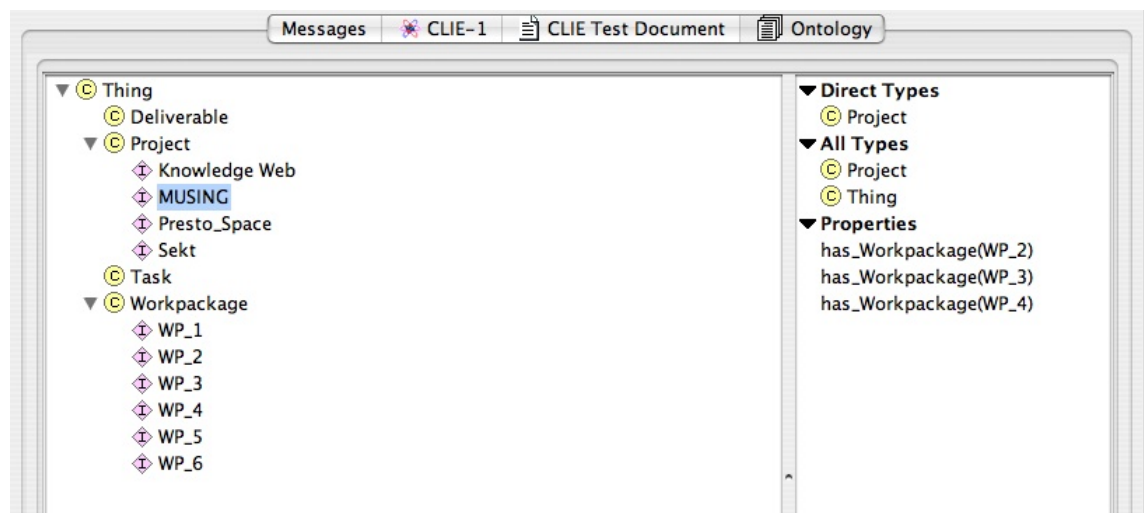
SEKT is a project. 'MUSING', 'Knowledge Web', and Presto Space are projects.

Projects have workpackages.
Workpackages can have tasks.
WP1, WP2, WP3, WP4, WP5 and WP6 are workpackages.

SEKT has WP1.

'MUSING' has WP2, WP3, and WP4.
'Knowledge Web' has WP5 and WP6.

Generates:



And output:

```
<#Project> rdf:type owl:Class .
<#Project> rdf:subClassOf <#Thing> .
```

```

<#Task> rdf:type owl:Class .
<#Task> rdf:subClassOf <#Thing> .
<#Workpackage> rdf:type owl:Class .
<#Workpackage> rdf:subClassOf <#Thing> .
<#Deliverable> rdf:type owl:Class .
<#Deliverable> rdf:subClassOf <#Thing> .
<#Sekt> rdf:type <#Project> .
<#Presto_Space> rdf:type <#Project> .
<#MUSING> rdf:type <#Project> .
<#Knowledge Web> rdf:type <#Project> .
<#has_Workpackage> rdf:type owl:ObjectProperty ;
    rdfs:domain <#Project> ;
    rdfs:range <#Workpackage> .
<#has_Task> rdf:type owl:ObjectProperty ;
    rdfs:domain <#Workpackage> ;
    rdfs:range <#Task> .
<#WP_1> rdf:type <#Workpackage> .
<#WP_3> rdf:type <#Workpackage> .
<#WP_4> rdf:type <#Workpackage> .
<#WP_6> rdf:type <#Workpackage> .
<#WP_2> rdf:type <#Workpackage> .
<#WP_5> rdf:type <#Workpackage> .
<#Sekt> <#has_Workpackage> <#WP_1> .
<#MUSING> <#has_Workpackage> <#WP_4> .
<#MUSING> <#has_Workpackage> <#WP_3> .
<#MUSING> <#has_Workpackage> <#WP_2> .
<#Knowledge Web> <#has_Workpackage> <#WP_5> .
<#Knowledge Web> <#has_Workpackage> <#WP_6> .

```

CLIE is not limited to one kind of discourse, and can cover many themes. For example text:

There are animals. There are limbs. Legs are a type of limb.
Paws are a type of leg.

Cats, birds, and fish are a type of animal.

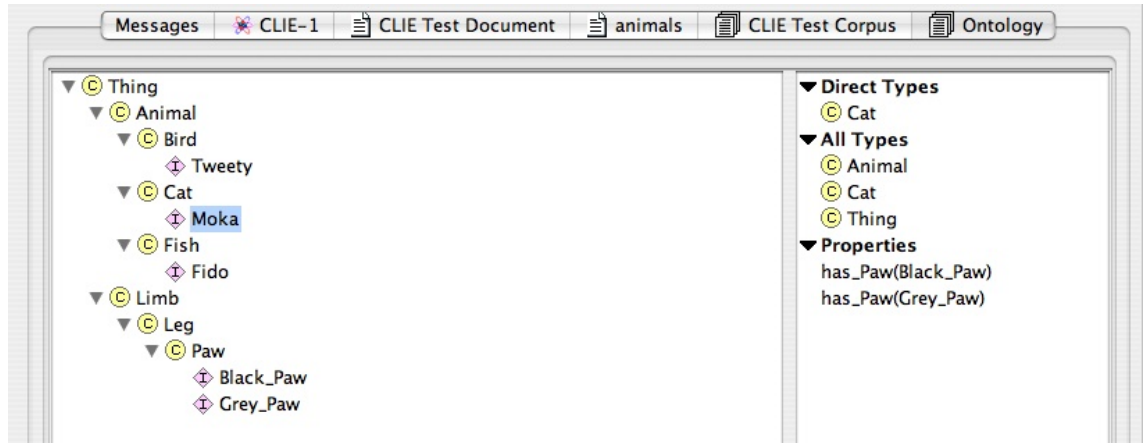
Cats have paws, whiskers, and tails. Fish have fins and scales.
Birds can have wings, feathers, and talons.

Moka is a cat. Tweety is a bird. Fido is a fish.

Grey paw and black paw are paws.

Moka has grey paw, and black paw.

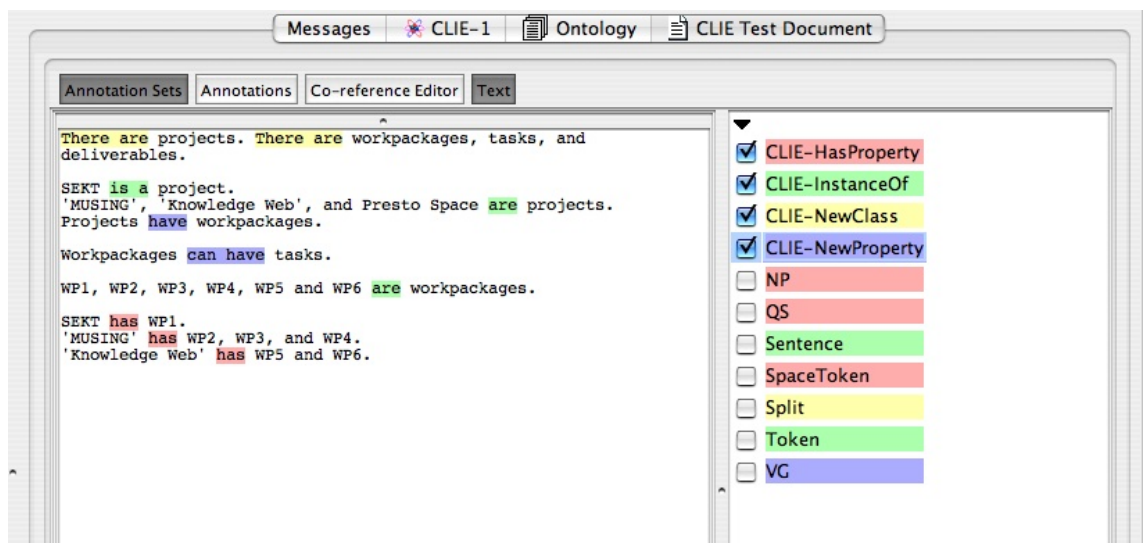
Generates:



5.3.2 How to use CLIE CL

CLIE CL can be considered a “fill in the blank” kind of controlled language. Certain sentence types only are considered and each of the sentences generates a deterministic result.

In the following example the input text is shown in GATE document editor. Once the application is run you can see the key phrases for each of the sentences. They are highlighted as GATE annotations



The following sentences can be used to generate triples:

Class existence

Sentence	Generates triple	Example
There are CLASS(es).	CLASS SubclassOf ROOT	There are projects.
There are CLASSs, CLASS2s, ..., and CLASSns.	CLASS1 SubclassOf Root, ... CLASSn SubclassOf ROOT	There are deliverables, work packages, and tasks.

Class hierarchy

SUBCLASS is a type of CLASS.	SUBCLASS SubclassOf CLASS	Cat is a type of animal.
SUBCLASS1, SUBCLASS2, ... and SUBCLASSn are a type of CLASS.	SUBCLASS1 SubclassOf CLASS, ..., SUBCLASSn SubclassOf CLASS	Wolves and foxes are a type of animal.

Instance of a Class

INSTANCE is a(n) CLASS.	INSTANCE InstanceOf CLASS	SEKT is a project.
INSTANCE1, INSTANCE2, ... and INSTANCEn are a(n) CLASS	INSTANCE1 InstanceOf CLASS, ..., INSTANCEn InstanceOf CLASS	WP1, WP2, WP3, are work packages.

Class and instance properties

CLASS (can) have PROPERTY.	CLASS CanHave PROPERTY	Projects can have work packages.
CLASS (can) have PROPERTY1, ... and PROPERTYn.	CLASS CanHave PROPERTY1, ..., CLASS CanHave PROPERTYn	Work packages can have deliverables and tasks.
INSTANCE has PROPERTY.	INSTANCE has PROPERTY	SEKT has WP1.
INSTANCE has PROPERTY1, ..., and PROPERTYn.	INSTANCE has PROPERTY1, ..., INSTANCE has PROPERTYn	SEKT has WP1, WP2 and WP3.

Further points about CLIE:

- Classes, instances, properties, values and other targets are assigned a standard format by CLIE unless they are surrounded by quotes. The standard CLIE label format is camel cased with underscores, i.e. *CLIE label* becomes:

Clie_Label

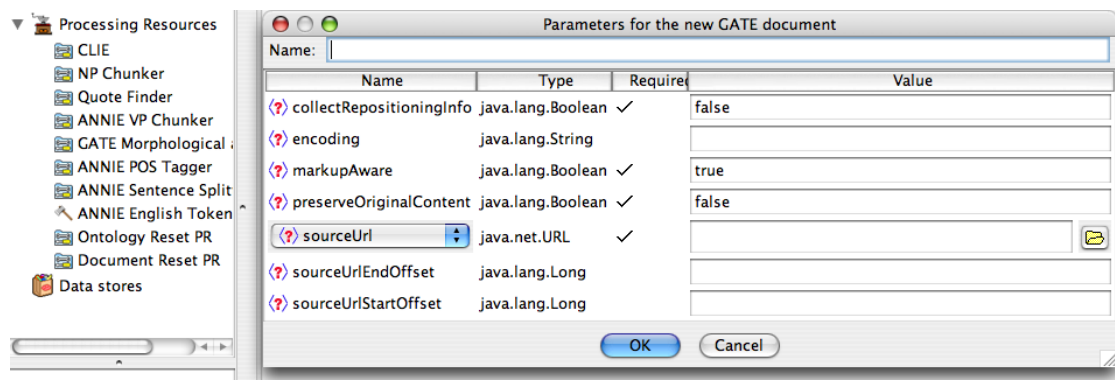
- Commas before *and* are optional, so both *WP1, WP2, and WP3* and *WP1, WP2 and WP3* are valid lists.

- When talking about an object in general plural can be used. For example, instead of saying *There are project.* or *There is a project.*, when trying to express the existence of projects in general, one can use the more natural form of *There are projects.*.
- Single quotes are used for preserving label orthography. In the above example we have '*MUSING*' in single quotes. This ensures that the word preserves its capitalisation and is not presented as a standard CLIE item name, in this case *Musing*.
- Double quotes are reserved for text descriptions.

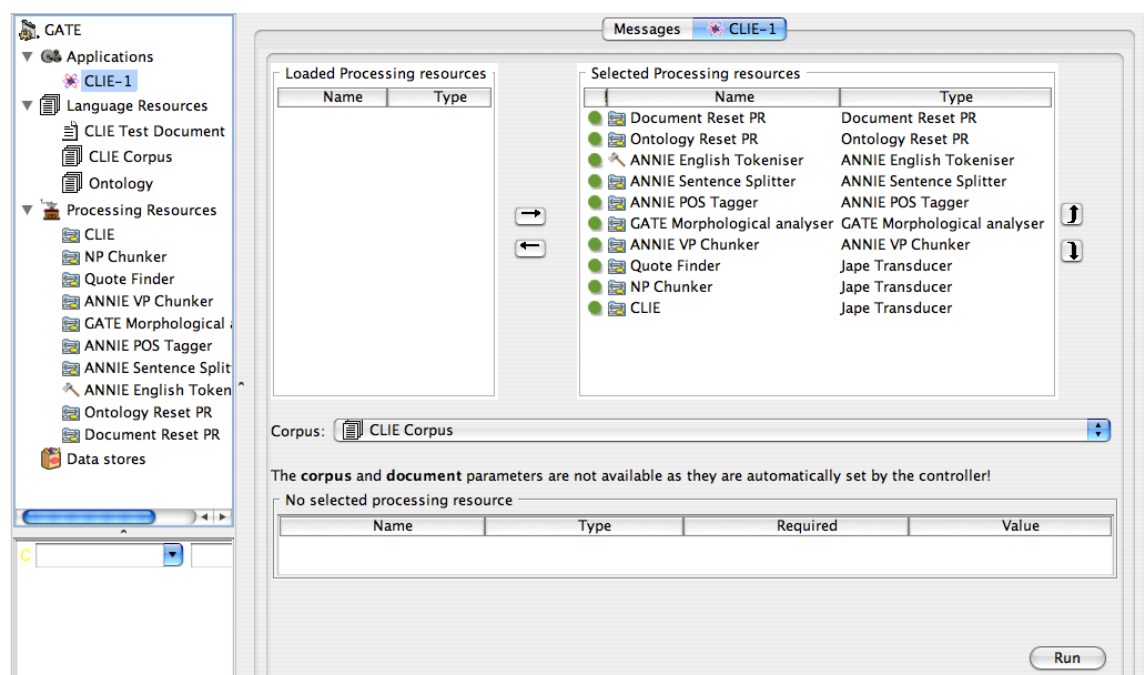
5.3.3 How to use CLIE through GATE

One way of using CLIE is through the GATE platform. This approach offers many advantages including an Ontology viewer, which visually represents the generated ontology, and annotations, which help guide the writing process.

1. You need to download GATE. GATE can be obtained from <http://gate.ac.uk/download>. Make sure you download GATE version 3.1-beta1 at least. The CLIE application will not work with GATE version 3.0 or earlier.
2. Refer to GATE documentation on the same site for instructions on how to run GATE for your platform.
3. Download the CLIE application from <http://gate.ac.uk/sale/gateplugins/clie/clie.zip> and unpack it to a directory on your computer. This will create a directory called "clie" which contains the application file called "clie.xgapp" as well as all the extra files required by the application.
4. Once GATE is downloaded and running go to the File menu and click on Restore Application from File. Browse to the location of clie.xgapp.
5. Load an existing corpus or create a new one.
 - (a) To load an existing corpus go to the File menu. Click on New language resource and then Gate corpus.
 - (b) To create a new corpus go to the File menu. Click on New language resource and then Gate document.
 - If you are opening text from an existing file specify the `sourceURL`.
 - Otherwise, change to `stringContent` and type in the data manually.
 - (c) The data can be changed and augmented during your GATE session, however this will not affect the original file from which the data was imported.



1. Add the corpus to the application. You can do this in the CLIE view which you can reach by double clicking on the CLIE application.
2. Run the application. You can either right click on the application in the left frame or click the Run button on the CLIE view.



1. CLIE will now populate the ontology and generate the annotations on the documents.
2. To view the annotations double-click on the document. Click on the button marked Annotation Sets, and expand the list which will show on the right. The CLIE specific annotations are prefixed with CLIE. CLIE-SentenceParsed annotation shows which sentences have been successfully translated into the ontology. CLIE-SentenceChunked shows which sentences were parsed using less rigorous parsing.

5.3.4 Design

The plain text input is processed using GATE [CMBT02]. The input is tokenised, tagged and analysed using a morphological analyser. The resulting annotations are processed with JAPE transducers and appropriate patterns are extracted.

The GATE architecture allows versatility, fast development, and direct interface with the ontologies. Using GATE the input text is processed with a tokeniser, tagger, and a morphological analyser. Nouns and specific noun phrases are identified using a JAPE transducer. Then another transducer searches for patterns over annotations looking for the types of sentences outlined above. In these sentences specific tokens and the marked nouns are used to extract information. The sentences are fully parsed which guarantees precision and no information loss.

The tokeniser separates input into tokens. It identifies words within a sentence and separates them from punctuation. For example in the sentence:

```
There are deliverables.
```

The tokens are:

```
[There] [are] [deliverables] [.]
```

The tagger finds the parts of speech for each of the tokens. In other words it finds out what kind of a word each of the tokens is; whether it is a noun, an adjective, a verb, etc.

```
[There]: existential quantifier
[are]: verb - 3rd person singular present
[deliverables]: noun - plural
```

The morphological analyser gives the roots of all the words.

```
[There]: root - there
[are]: root - be
[deliverables]: root - deliverable
```

The morphological analyser allows these general types of sentences which announce existence of classes without the need for using artificial singular expressions, i.e. *There is deliverable*.

The first JAPE transducer will take the above annotated sentence and look for and mark noun patterns which are likely candidates for CLASS, INSTANCE, and other ontological objects.

The second JAPE transducer, and the final step, is CLIE itself. It looks for specific patterns outlined above to extract the information. For example:

There are ----.

It then calls Clie.java class function:

```
Clie.addTriple(Ontology, Subject, Object, Predicate)
```

The Clie class mediates between the textual input and the Ontology.

5.4 Summary and Future Work

The controlled language IE task has developed a simplified natural language processor that, on the one hand, allows the specification of logical data for Semantic KT purposes in normal language, while, on the other hand, attaining the high accuracy levels necessary for high-reliability applications. Therefore, users now have alternative means for building ontologies, other than the ‘traditional’ ontology editors such as Protege.

CLIE is based on GATE’s existing FST cascade IE [CMBT02]. CLIE is configured so that it either accepts input as valid (in which case accuracy is in general 100%) or rejects it and warns the user of needed repairs to their syntax. In certain cases the system will try a less strict analysis mode in order to suggest how repair may be effected. Therefore, because the parsing process is deterministic, accuracy is not really an issue.

In year 3 work will focus on evaluating CLIE and extending it to cover a richer set of language constructs.

Bibliography

- [AS92] G. Adriaens and D. Schreurs. From COGRAM to ALCOGRAM: Toward a controlled English Grammar Checker. In *Conference on Computational Linguistics (COLING'92)*, pages 595–601, Nantes, France, 1992.
- [CMBT02] 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.
- [KAMN98] C. Kamprath, E. Adolphson, T. Mitamura, and E. Nyberg. Controlled Language for Multilingual Document Production: Experience with Caterpillar Technical English. In *Second International Workshop on Controlled Language Applications (CLAW '98)*, 1998.
- [PM03] A. Pease and W. Murray. An english to logic traslator for ontology-based knowledge representation languages. In *In Proceedings of the 2003 IEEE International Conference on Natural Language Processing and Knowledge Engineering*, pages 777–783, Beijing, China, 2003.
- [Pul96] S. Pulman. Controlled Language for Knowledge Representation. In *CLAW96: Proceedings of the First International Workshop on Controlled Language Applications*, pages 233–242, Leuven, Belgium, 1996.
- [Sch00] Uta Schwertel. Controlling plural ambiguities in Attempto Controlled English. In *Proceedings of the 3rd International Workshop on Controlled Language Applications*, Seattle, Washington, 2000.
- [Sku03] D. Skuce. A Controlled Language for Knowledge Formulation on the Semantic Web. <http://www.site.uottawa.ca:4321/factguru2.pdf>, 2003.
- [Sow02] J. Sowa. Architectures for intelligent systems. *IBM Systems Journal*, 41(3), 2002.

Appendix A

Yam Syntax

A.1 Overview of JAPE

This section describes briefly JAPE – a Java Annotation Patterns Engine¹. JAPE provides finite state transduction over annotations based on regular expressions. JAPE is a version of CPSL – Common Pattern Specification Language².

JAPE allows you to recognise regular expressions in annotations on documents. A JAPE grammar consists of a set of phases, each of which consists of a set of pattern/action rules. The phases run sequentially and constitute a cascade of finite state transducers over annotations. The left-hand-side (LHS) of the rules consist of an annotation pattern that may contain regular expression operators (e.g. `*`, `?`, `+`). The right-hand-side (RHS) consists of annotation manipulation statements. Annotations matched on the LHS of a rule may be referred to on the RHS by means of labels that are attached to pattern elements.

At the beginning of each grammar, several options can be set:

- Control - this defines the method of rule matching
- Debug - this enables debugging of JAPE grammars - see <http://gate.ac.uk/sale/tao/index.html#chap:jape>.

Input annotations must also be defined at the start of each grammar. If no annotations are defined, the default will be Token, SpaceToken and Lookup (i.e. only these annotations will be considered when attempting a match).

There are 3 main ways in which the pattern can be specified:

¹For a complete introduction see <http://gate.ac.uk/sale/tao/index.html#chap:jape>.

²A good description of the original version of this language is in <http://www.ai.sri.com/~appelt/TextPro> Doug Appelt's TextPro manual. Doug was a great help to us in implementing JAPE. Thanks Doug!

- specify a string of text, e.g. `{Token.string == "of"}`
- specify an annotation previously assigned from a gazetteer, tokeniser, or other module, e.g. `{Lookup}`
- specify the attributes (and values) of an annotation), e.g. `{Token.kind == number}`

Macros can also be used in the LHS of rules. This means that instead of expressing the information in the rule, it is specified in a macro, which can then be called in the rule. The reason for this is simply to avoid having to repeat the same information in several rules. Macros can themselves be used inside other macros.

The same operators can be used as for the tokeniser rules, i.e.

```
|
*
?
+
```

The pattern description is followed by a label for the annotation. A label is denoted by a preceding semi-colon; in the example below, the label is `:location`.

The RHS of the rule contains information about the annotation. Information about the annotation is transferred from the LHS of the rule using the label just described, and annotated with the entity type (which follows it). Finally, attributes and their corresponding values are added to the annotation. Alternatively, the RHS of the rule can contain Java code to create or manipulate annotations.

In the simple example below, the pattern described will be awarded an annotation of type “Enamex” (because it is an entity name). This annotation will have the attribute “kind”, with value “location”, and the attribute “rule”, with value “GazLocation”. (The purpose of the “rule” attribute is simply to ease the process of manual rule validation).

```
Rule: GazLocation
(
{Lookup.majorType == location}
)
:location -->
:location.Enamex = {kind="location", rule=GazLocation}
```

It is also possible to have more than one pattern and corresponding action, as shown in the rule below. On the LHS, each pattern is enclosed in a set of round brackets and has a unique label; on the RHS, each label is associated with an action. In this example, the Lookup annotation is labelled “jobtitle” and is given the new annotation JobTitle; the TempPerson annotation is labelled “person” and is given the new annotation “Person”.

```

Rule: PersonJobTitle
Priority: 20

(
  {Lookup.majorType == jobtitle}
):jobtitle
(
  {TempPerson}
):person
-->
  :jobtitle.JobTitle = {rule = "PersonJobTitle"},
  :person.Person = {kind = "personName", rule = "PersonJobTitle"}

```

Similarly, labelled patterns can be nested, as in the example below, where the whole pattern is annotated as `Person`, but within the pattern, the `jobtitle` is annotated as `JobTitle`.

```

Rule: PersonJobTitle2
Priority: 20

(
  (
    {Lookup.majorType == jobtitle}
  ):jobtitle
  {TempPerson}
):person
-->
  :jobtitle.JobTitle = {rule = "PersonJobTitle"},
  :person.Person = {kind = "personName", rule = "PersonJobTitle"}

```

A.2 Yam Syntax

A.2.1 Bold, italic and teletype

Bold text is contained in stars: `*this is bold*` becomes **this is bold**.

Italic text is contained in underscores: `_this is italic_` becomes *this is italic*.

Fixed-width text is contained in equals signs: `=this is teletype=` becomes this is teletype.

A.2.2 Horizontal lines

Horizontal lines are indicated by 2 or more For example:

%%

and

%%

both result in:

A.2.3 Lists

Unordered lists are indicated by 'o' at the start of a line, and ordered lists by '-'. Nesting is indicated by two spaces preceding the item indicator. For example:

```
- This is an unordered list
- Second item
  o This is a nested...
  o ...ordered list
- Back to the third item of the enclosing list
```

results in:

- This is an unordered list
 - Second item
 1. This is a nested...
 2. ...ordered list
 - Back to the third item of the enclosing list
-

A.2.4 Verbatim output

Verbatim output starts with '<<' at the start of a line and ends with '>>'. For example:

```
<<
This *will not* get translated
>>
```

When the target language is HTML, for example, the output will contain

```
<pre>
```

tags. It is also possible to tell the translator to write output directly without any intervention, using << >> :

```
<pre> This will not get translated either, but any markup in the target language will be
interpreted in that language. ) </pre>
```

A.2.5 Notes

Notes are like this:

```
%notes("This is a note")
```

The contents will be output to the translation file, but will be commented out in that file. The quotation marks around the note are necessary; notes cannot contain quotation marks (even if escaped).

A.2.6 Escapes

To stop a special character from being interpreted, use a `'\'`. For example,

```
\%%
```

will not generate a line.

Some syntax elements interact with each other and produce unexpected escaping behaviour. For example, in

```
'=http://gate.ac.uk/='
```

the equals signs are translated, but not the URL they contain (with the result `'http://gate.ac.uk/')`.

A.2.7 Headings

Headings are lines starting with %1 (for first level), %2, %3 or %4. For example, the heading for this section is

```
%1 Headings
```

A.2.8 Links and anchors

Links can be specified in four ways:

1. As plain text, e.g. `'http://gate.ac.uk/'` will become `http://gate.ac.uk/`

2. Using `'%(target)'`, e.g. `%(http://gate.ac.uk/)` will become `http://gate.ac.uk/`
3. Using `'%(target, label)'`, e.g. `%(http://gate.ac.uk/, GATE home)` will become `GATE home`
4. Using Wiki syntax

anchors and labels are specified using `'#name'`. For example,

```
%2 A Heading #label
```

will result in a heading followed by the anchor `label`.

A.2.9 Quotations

Quotations are enclosed in `"` marks that are preceded by two spaces at the start of a line. For example,

```
"This is a quote"
```

becomes:

```
  This is a quote
```

A.2.10 Line breaks

Line breaks are indicated by a backslash at the end of a line. For example:

```
This line is broken\  
in two.
```

becomes:

```
This line is broken  
in two.
```

A.2.11 Tables

Tables use square brackets, bars and dashes. For example:

```
%[
*header col 1*          | *header col 2*
----
row 1 col 1             | col 2
----
row 2 col 1             | col 2
%]
```

results in:

header col 1	header col 2
row 1 col 1	col 2
row 2 col 1	col 2

A.3 YAM syntax design

A.3.1 Bugs:

1. include plugin only works from command line, otherwise it looks for the file in the plugins directory, and that's no good. get resource doesn't appear to work well either in this case
2. the target in a url gets interpreted as an anchor when the protocol isn't specified (e.g. `antlr.org/doc/lexer.html#unicode`)
3. unclosed = causes open `tt` with no close
4. text at the end of a URL can get included in the URL, e.g. `http://antlr.org/doc/lexer.html#unicode:` includes the ":"
5. there's no way to end an embedded list element except by another list element (embedded or higher level)

Completed:

1. *the citation plugin is HTML specific*
2. *the citation plugin closes any embedding lists; what is needed is to be able to tell the context not to do further processing on the results of the plugin, instead of using the output mechanism*

3. *an empty notes field results in null pointer exception*

A.3.2 Wish list:

- anchors that are at the end of a heading line should be sent to the translator as a separate call, `link(url, title, anchor)`, so that the translator can position the anchor before or after the heading as appropriate
- allow `_` to be escaped within an emphasised phrase
- allow a string of percents at the end of a line to be a comment
- section level one should translate to H1
- WikiLinks (see below)
- definition lists like twiki
- variables, e.g. like twiki's null
- table summary attributes
- UTF-8

Completed:

- *"generated file" warning in the output file*
- *autogenerate anchor/label from normalised words of header + int?*
 1. this is only done when building the table of contents, but can be easily changed to work otherwise too.
 2. the anchors are the same as the number preceding the heading, e.g. "1.1."
- *citation*
- *auto parsing all in-line links like http:, mailto:, ...*
- *ability to create mailto and ftp links with text - %(mailto:...) or %(mailto:..., ...)*
 1. anything that's within `%()` is a link, regardless of it's format
- *auto-numbering of sections in the HTML translator*
- *images*
- *%contents with numbered links to sections*

- *double dashes: – makes a long dash*
- *get escaping of etc. to works except within the markup itself,i.e. I couldn't escape an underscore in this sentence*
- *allow - in title*
- *allow e.g. - in anchors*

A.3.3 WikiLinks

WikiLinks are just links, created either like other links (`% (. . .)`), or by typing a Wiki-Word. Some points:

- WikiSyntaxForLinks should include an optional relative path, e.g. `path/to/WikiWord`
- when the target of a WikiLink exists, it is treated exactly like a normal link
- when the target doesn't exist, the link is rendered specially, and clicking directs you to a `create` page
- therefore, the renderer (`yam2 . . .`) has to know whether a local link exists or not in every case

A.4 CLIE implementation

Main translation and parsing is done through the JAPE files.

A.4.1 `clie.jape`

This file does main CLIE translation.

```
/*
 * clie.jape
 *
 * Copyright (c) 1998-2005, The University of Sheffield.
 *
 */
```

Phase: CLIE
 Input: Token NP QS Split
 Options: control = appelt

```
Macro:String
(
    ({Token.string==" "}{QS}):stringItem{Token.string==" "})
)
```

```
Macro:NP
(
    ({NP}):items |
    ({Token.string=="' "}{QS}):verbItem{Token.string=="' "})
)
```

```
Macro:ObjectNP
(
    ({NP}):object |
    ({Token.string=="' "}{QS}):verbObject{Token.string=="' "})
)
```

```
Macro:PropertyNP
(
    ({NP}):property |
    ({Token.string=="' "}{QS}):verbProperty{Token.string=="' "})
)
```

```
Macro:Chunk
(
    (({Token.kind==word}|{Token.kind==number}+):items |
    ({Token.string=="' "}{QS}):verbItem{Token.string=="' "})
)
```

```
Macro:NPList
(
    ((NP){Token.string==" , "}) *
    (NP)
    ({Token.string==" , "}) ?
    {Token.string=="and"}
    (NP)
)
```

```
Macro:ChunkList
(
    ((Chunk){Token.string==" , "}) *
```

```

    (Chunk)
    ({Token.string=="", ""})?
        {Token.string=="and"}
    (Chunk)
)

```

Rule: There_are_objects

```

((({Token.string == "there"}|{Token.string == "There"})
    ({Token.string == "are"}|{Token.string == "is"})):keyphrase
    (NPList|NP)
    {Split}
):parsed
-->
{
    Annotation tokAnn;
    Iterator objIter;
    String object = "";

    Annotation objAnn;
    List tokens;

    AnnotationSet objects = (AnnotationSet)bindings.get("items");
    if(objects != null) {
        objIter = objects.iterator();

        while(objIter.hasNext()) {
            object = "";
            objAnn = (Annotation)objIter.next();
            tokens = new ArrayList(inputAS.get("Token",
                objAnn.getStartNode().getOffset(),
                objAnn.getEndNode().getOffset()));
            Collections.sort(tokens, new OffsetComparator());
            for(int i = 0; i < tokens.size() -1; i++){
                tokAnn = (Annotation)tokens.get(i);
                object +=
                    sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                        get("string"))+ "_";
            }
            tokAnn = (Annotation)tokens.get(tokens.size() -1);
            object +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                    get("root"));
        }
    }
}

```

```

        sins.clie.Clie.newClass(ontology, object, sins.clie.Clie.
            TOP_CLASS_NAME);
    }
}

objects = (AnnotationSet)bindings.get("verbItem");
if(objects != null) {
    objIter = objects.iterator();

    while(objIter.hasNext()) {
        object = "";
        objAnn = (Annotation)objIter.next();
        object = (String)objAnn.getFeatures().get("string");
        sins.clie.Clie.newClass(ontology, object, sins.clie.Clie.
            TOP_CLASS_NAME);
    }
}

AnnotationSet parsedAS = (AnnotationSet)bindings.get("parsed");
AnnotationSet keyPhraseAS = (AnnotationSet)bindings.
    get("keyphrase");
FeatureMap features = Factory.newFeatureMap();
features.put("rule", "There_are_objects");
try {
    outputAS.add(keyPhraseAS.firstNode().getOffset(),
        keyPhraseAS.lastNode().getOffset(),
        "CLIE-NewClass", features);
    outputAS.add(parsedAS.firstNode().getOffset(),
        parsedAS.lastNode().getOffset(),
        "CLIE-SentenceParsed", Factory.newFeatureMap());
} catch (Exception ioe) {
    ioe.printStackTrace();
}
}

```

Rule: Thing_is_an_object

```

((NP|NPList)
    (
        ({Token.string == "is"})
        ({Token.string == "a"} | {Token.string == "an"})
        |
        {Token.string == "are"}
    )
)

```

```

        ):keyphrase
        ObjectNP
        {Split}
):parsed
-->
{
    Annotation tokAnn;
    Iterator thingIter;
    List tokens;
    String object = "";
    String thing = "";
    if(bindings.get("object") != null) {
        Annotation objAnn =
            (Annotation)((AnnotationSet)bindings.get("object")).
                iterator().next();
        //input annotations are in inputAS
        tokens = new ArrayList(inputAS.get("Token",
            objAnn.getStartNode().getOffset(),
            objAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());

        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);
            object +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                    get("string")) + "_";
        }
        tokAnn = (Annotation)tokens.get(tokens.size() -1);
        object +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("root"));
    }
    else if(bindings.get("verbObject") != null) {
        Annotation objAnn =
            (Annotation)((AnnotationSet)bindings.get("verbObject")).
                iterator().next();
        //input annotations are in inputAS

        object += (String)objAnn.getFeatures().get("string");
    }

    AnnotationSet things = (AnnotationSet)bindings.get("items");

```

```

if( things != null) {

    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        thing = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        tokens = new ArrayList(inputAS.get("Token",
            thingAnn.getStartNode().getOffset(),
            thingAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());
        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);
            thing +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                    get("string")) + "_";
        }
        tokAnn = (Annotation)tokens.get(tokens.size() -1);
        thing +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("root"));

        sins.clie.Clie.hasInstance(ontology, object, thing);
    }
}

things = (AnnotationSet)bindings.get("verbItem");

if( things != null) {
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        thing = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        thing = (String)thingAnn.getFeatures().get("string");
        sins.clie.Clie.hasInstance(ontology, object, thing);
    }
}

AnnotationSet parsedAS = (AnnotationSet)bindings.get("parsed");
AnnotationSet keyPhraseAS = (AnnotationSet)bindings.
    get("keyphrase");
FeatureMap features = Factory.newFeatureMap();
features.put("rule", "Thing_is_an_object");
try {

```



```

        outputAS.add(keyPhraseAS.firstNode().getOffset(),
            keyPhraseAS.lastNode().getOffset(),
            "CLIE-InstanceOf", features);
        outputAS.add(parsedAS.firstNode().getOffset(),
            parsedAS.lastNode().getOffset(),
            "CLIE-SentenceParsed", Factory.newFeatureMap());
    } catch (Exception ioe) {
        ioe.printStackTrace();
    }
}
Rule: Thing_is_a_type_of_object

(
    (NP|NPList)
    (
        ({Token.string == "is"}|{Token.string == "are"})
        ({Token.string == "a"})
        {Token.string == "type"}
        {Token.string == "of"}
    ):keyphrase
    ObjectNP
    {Split}
):parsed
-->
{
    Annotation tokAnn;
    Iterator thingIter;
    List tokens;
    String object = "";
    String thing = "";

    if(bindings.get("object") != null) {
        Annotation objAnn =
            (Annotation)((AnnotationSet)bindings.get("object")).
                iterator().next();
        //input annotations are in inputAS
        tokens = new ArrayList(inputAS.get("Token",
            objAnn.getStartNode().getOffset(),
            objAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());

        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);

```

```

        object +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("string")) + "_";
    }
    tokAnn = (Annotation)tokens.get(tokens.size() -1);
    object +=
        sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
            get("root"));
    }
else if(bindings.get("verbObject") != null) {
    Annotation objAnn =
        (Annotation)((AnnotationSet)bindings.get("verbObject")).
            iterator().next();
    //input annotations are in inputAS

    object += (String)objAnn.getFeatures().get("string");

}

if( bindings.get("items") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("items");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        thing = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        tokens = new ArrayList(inputAS.get("Token",
            thingAnn.getStartNode().getOffset(),
            thingAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());
        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);
            thing +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                    get("string")) + "_";
        }
        tokAnn = (Annotation)tokens.get(tokens.size() -1);
        thing +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("root"));
        sins.clie.Clie.newClass(ontology, thing, object);
    }
}

```

```

    }
}
if( bindings.get("verbItem") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("verbItem");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        thing = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        thing = (String)thingAnn.getFeatures().get("string");

        sins.clie.Clie.newClass(ontology, thing, object);

    }
}
AnnotationSet parsedAS = (AnnotationSet)bindings.get("parsed");
AnnotationSet keyPhraseAS = (AnnotationSet)bindings.
get("keyphrase");
FeatureMap features = Factory.newFeatureMap();
features.put("rule", "Thing_is_a_type_of_object");
try {
    outputAS.add(keyPhraseAS.firstNode().getOffset(),
        keyPhraseAS.lastNode().getOffset(),
        "CLIE-Subclass", features);
    outputAS.add(parsedAS.firstNode().getOffset(),
        parsedAS.lastNode().getOffset(),
        "CLIE-SentenceParsed", Factory.newFeatureMap());
} catch (Exception ioe) {
    ioe.printStackTrace();
}
}

Rule: Object_can_have_property

(
    ObjectNP
    (({Token.string == "can"})?
        {Token.string == "have"}):keyphrase
        (NPList|NP)
        {Split}
):parsed

```

```

-->
{
    Annotation tokAnn;
    Iterator thingIter;
    List tokens;
    String object = "";
    String property = "";
    if(bindings.get("object") != null) {
        Annotation objAnn =
            (Annotation)((AnnotationSet)bindings.get("object")).
                iterator().next();
        //input annotations are in inputAS
        tokens = new ArrayList(inputAS.get("Token",
            objAnn.getStartNode().getOffset(),
            objAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());

        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);
            object +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                    get("string")) + "_";
        }
        tokAnn = (Annotation)tokens.get(tokens.size() -1);
        object +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("root"));
    }
    else if(bindings.get("verbObject") != null) {
        Annotation objAnn =
            (Annotation)((AnnotationSet)bindings.get("verbObject")).
                iterator().next();
        //input annotations are in inputAS

        object += (String)objAnn.getFeatures().get("string");

    }
    if( bindings.get("items") != null) {
        AnnotationSet things = (AnnotationSet)bindings.get("items");
        thingIter = things.iterator();

        while(thingIter.hasNext()) {

```

```

    property = "";
    Annotation thingAnn = (Annotation)thingIter.next();
    tokens = new ArrayList(inputAS.get("Token",
        thingAnn.getStartNode().getOffset(),
        thingAnn.getEndNode().getOffset()));
    Collections.sort(tokens, new OffsetComparator());
    for(int i = 0; i < tokens.size() -1; i++){
        tokAnn = (Annotation)tokens.get(i);
        property +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("string")) + "_";
    }
    tokAnn = (Annotation)tokens.get(tokens.size() -1);
    property +=
        sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
            get("root"));

    sins.clie.Clie.hasObjectProperty(ontology, object, "has_"+
        property, property);
}
}
if( bindings.get("verbItem") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("verbItem");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        property = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        property = (String)thingAnn.getFeatures().get("string");

        sins.clie.Clie.hasObjectProperty(ontology, object,
            "has_"+property, property);
    }
}
AnnotationSet parsedAS = (AnnotationSet)bindings.get("parsed");
AnnotationSet keyPhraseAS = (AnnotationSet)bindings.
    get("keyphrase");
FeatureMap features = Factory.newFeatureMap();
features.put("rule", "Object_can_have_property");
try {
    outputAS.add(keyPhraseAS.firstNode().getOffset(),
        keyPhraseAS.lastNode().getOffset(),
        "CLIE-NewProperty", features);
}

```

```

        outputAS.add(parsedAS.firstNode().getOffset(),
                     parsedAS.lastNode().getOffset(), "CLIE-SentenceParsed",
                     Factory.newFeatureMap());
    } catch (Exception ioe) {
        ioe.printStackTrace();
    }
}

```

Rule: Object_has_property

```

(ObjectNP
    ({Token.string == "has"}):keyphrase
    (NPList|NP)
    {Split}
):parsed
-->
{
    Annotation tokAnn;
    Iterator thingIter;
    List tokens;
    String object = "";
    String property = "";

    if(bindings.get("object") != null) {
        Annotation objAnn =
            (Annotation)((AnnotationSet)bindings.get("object")).
            iterator().next();
        //input annotations are in inputAS
        tokens = new ArrayList(inputAS.get("Token",
            objAnn.getStartNode().getOffset(),
            objAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());

        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);
            object +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("string")) + "_";
        }
        tokAnn = (Annotation)tokens.get(tokens.size() -1);
        object +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
            get("root"));
    }
}

```

```

}
else if(bindings.get("verbObject") != null) {
    Annotation objAnn =
        (Annotation)((AnnotationSet)bindings.get("verbObject")).
        iterator().next();
    //input annotations are in inputAS

    object += (String)objAnn.getFeatures().get("string");

}
if( bindings.get("items") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("items");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        property = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        tokens = new ArrayList(inputAS.get("Token",
            thingAnn.getStartNode().getOffset(),
            thingAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());
        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);
            property +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("string")) + "_";
        }
        tokAnn = (Annotation)tokens.get(tokens.size() -1);
        property +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
            get("root"));

        sins.clie.Clie.hasPropertyValue(ontology, object, property);
    }
}
if( bindings.get("verbItem") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("verbItem");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        property = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        property = (String)thingAnn.getFeatures().get("string");
    }
}

```

```

        sins.clie.Clie.hasPropertyValue(ontology, object, property);
    }
}

AnnotationSet parsedAS = (AnnotationSet)bindings.get("parsed");
AnnotationSet keyPhraseAS = (AnnotationSet)bindings.
get("keyphrase");
FeatureMap features = Factory.newFeatureMap();
features.put("rule", "Object_has_property");
try {
    outputAS.add(keyPhraseAS.firstChild().getOffset(),
        keyPhraseAS.lastNode().getOffset(),
        "CLIE-HasProperty", features);
    outputAS.add(parsedAS.firstChild().getOffset(),
        parsedAS.lastNode().getOffset(),
        "CLIE-SentenceParsed",
        Factory.newFeatureMap());
} catch (Exception ioe) {
    ioe.printStackTrace();
}

}

Rule: Object_has_property_with_value_value_NP
Priority:50
(
    ObjectNP
    ({Token.string == "has"}|{Token.string == "have"})
    PropertyNP
    ({Token.string == "with"}
    {Token.string == "value"} |
    {Token.string == "which"}
    ({Token.string == "is"}|{Token.string == "are"})):keyphrase
    (NPList|NP|String)
    {Split}
):parsed
-->
{
    Annotation tokAnn;
    Iterator thingIter;

```



```

List tokens;
String object = "";
String property = "";
String value = "";
if(bindings.get("object") != null) {
    Annotation objAnn =
        (Annotation)((AnnotationSet)bindings.get("object")).
            iterator().next();
    //input annotations are in inputAS
    tokens = new ArrayList(inputAS.get("Token",
        objAnn.getStartNode().getOffset(),
        objAnn.getEndNode().getOffset()));
    Collections.sort(tokens, new OffsetComparator());

    for(int i = 0; i < tokens.size() -1; i++){
        tokAnn = (Annotation)tokens.get(i);
        object +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("string")) + "_";
    }
    tokAnn = (Annotation)tokens.get(tokens.size() -1);
    object +=
        sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
            get("root"));
}
else if(bindings.get("verbObject") != null) {
    Annotation objAnn =
        (Annotation)((AnnotationSet)bindings.get("verbObject")).
            iterator().next();
    //input annotations are in inputAS

    object += (String)objAnn.getFeatures().get("string");

}
if( bindings.get("property") != null) {
    Annotation thingAnn =
        (Annotation)((AnnotationSet)bindings.get("property")).
            iterator().next();
    property = "";
    //input annotations are in inputAS
    tokens = new ArrayList(inputAS.get("Token",
        thingAnn.getStartNode().getOffset(),

```

```

        thingAnn.getEndNode().getOffset());
Collections.sort(tokens, new OffsetComparator());
for(int i = 0; i < tokens.size() -1; i++){
    tokAnn = (Annotation)tokens.get(i);
    property +=
        sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
            get("string")) + "_";
}
tokAnn = (Annotation)tokens.get(tokens.size() -1);
property +=
    sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
        get("root"));
property = "has_" + property;
}
else if( bindings.get("verbProperty") != null) {
    Annotation thingAnn =
        (Annotation)((AnnotationSet)bindings.get("verbProperty")).
            iterator().next();

    property = (String)thingAnn.getFeatures().get("string");

    property = "has_" + property;
}
if( bindings.get("items") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("items");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        value = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        tokens = new ArrayList(inputAS.get("Token",
            thingAnn.getStartNode().getOffset(),
            thingAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());
        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);
            value +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                    get("string")) + "_";
        }
        tokAnn = (Annotation)tokens.get(tokens.size() -1);
        value +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().

```

```

        get("root"));
        sins.clie.Clie.hasPropertyValue(ontology, object, property,
        value);
    }
}
if( bindings.get("verbItem") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("verbItem");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        value = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        value  = (String)thingAnn.getFeatures().get("string");

        sins.clie.Clie.hasPropertyValue(ontology, object, property,
        value);
    }

}
if( bindings.get("stringItem") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("stringItem");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        value = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        value  = (String)thingAnn.getFeatures().get("string");

        sins.clie.Clie.hasPropertyTextValue(ontology, object,
        property, value);
    }
}
AnnotationSet parsedAS = (AnnotationSet)bindings.get("parsed");
AnnotationSet keyPhraseAS = (AnnotationSet)bindings.
get("keyphrase");
FeatureMap features = Factory.newFeatureMap();
features.put("rule", "Object_has_property_with_value_value_NP");
try {
    outputAS.add(keyPhraseAS.firstNode().getOffset(),
        keyPhraseAS.lastNode().getOffset(),
        "CLIE-PropertyValue", features);
    outputAS.add(parsedAS.firstNode().getOffset(),
        parsedAS.lastNode().getOffset(),

```

```

        "CLIE-SentenceParsed",
        Factory.newFeatureMap());
    } catch (Exception ioe) {
        ioe.printStackTrace();
    }
}

Rule: Object_has_property_with_value_value_CHUNK
Priority:5
(
    ObjectNP
    ({Token.string == "has"}|{Token.string == "have"})
    PropertyNP
    ({Token.string == "with"}
    {Token.string == "value"} |
    {Token.string == "which"}
    ({Token.string == "is"}|{Token.string == "are"})):keyphrase
    (ChunkList|Chunk)
    {Split}
):parsed
-->
{
    Annotation tokAnn;
    Iterator thingIter;
    List tokens;
    String object = "";
    String property = "";
    String value = "";
    if(bindings.get("object") != null) {
        Annotation objAnn =
            (Annotation)((AnnotationSet)bindings.get("object")).
            iterator().next();
        //input annotations are in inputAS
        tokens = new ArrayList(inputAS.get("Token",
            objAnn.getStartNode().getOffset(),
            objAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());

        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);
            object +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("string")) + "_";
        }
    }
}

```

```

    }
    tokAnn = (Annotation)tokens.get(tokens.size() -1);
    object +=
        sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
            get("root"));
}
else if(bindings.get("verbObject") != null) {
    Annotation objAnn =
        (Annotation)((AnnotationSet)bindings.get("verbObject")).
            iterator().next();
    //input annotations are in inputAS

    object += (String)objAnn.getFeatures().get("string");
}

if( bindings.get("property") != null) {
    Annotation thingAnn =
        (Annotation)((AnnotationSet)bindings.get("property")).
            iterator().next();
    property = "";
    //input annotations are in inputAS
    tokens = new ArrayList(inputAS.get("Token",
        thingAnn.getStartNode().getOffset(),
        thingAnn.getEndNode().getOffset()));
    Collections.sort(tokens, new OffsetComparator());
    for(int i = 0; i < tokens.size() -1; i++){
        tokAnn = (Annotation)tokens.get(i);
        property +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("string")) + "_";
    }
    tokAnn = (Annotation)tokens.get(tokens.size() -1);
    property +=
        sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
            get("root"));
    property = "has_" + property;
}
else if( bindings.get("verbProperty") != null) {
    Annotation thingAnn =
        (Annotation)((AnnotationSet)bindings.get("verbProperty")).
            iterator().next();

```

```

property = (String)thingAnn.getFeatures().get("string");

property = "has_" + property;
}
if( bindings.get("items") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("items");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        value = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        tokens = new ArrayList(inputAS.get("Token",
            thingAnn.getStartNode().getOffset(),
            thingAnn.getEndNode().getOffset()));
        Collections.sort(tokens, new OffsetComparator());
        for(int i = 0; i < tokens.size() -1; i++){
            tokAnn = (Annotation)tokens.get(i);
            value +=
                sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                    get("string")) + "_";
        }
        tokAnn = (Annotation)tokens.get(tokens.size() -1);
        value +=
            sins.clie.Clie.camelCase((String)tokAnn.getFeatures().
                get("root"));
        sins.clie.Clie.hasPropertyValue(ontology, object, property,
            value);
    }
}
if( bindings.get("verbItem") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("verbItem");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        value= "";
        Annotation thingAnn = (Annotation)thingIter.next();
        value = (String)thingAnn.getFeatures().get("string");
        sins.clie.Clie.hasPropertyValue(ontology, object, property,
            value);
    }
}
}

```

```

if( bindings.get("stringtem") != null) {
    AnnotationSet things = (AnnotationSet)bindings.get("stringItem");
    thingIter = things.iterator();

    while(thingIter.hasNext()) {
        value = "";
        Annotation thingAnn = (Annotation)thingIter.next();
        value = (String)thingAnn.getFeatures().get("string");

        sins.clie.Clie.hasPropertyTextValue(ontology, object, property,
        value);
    }
}
AnnotationSet parsedAS = (AnnotationSet)bindings.get("parsed");
AnnotationSet keyPhraseAS = (AnnotationSet)bindings.
get("keyphrase");
FeatureMap features = Factory.newFeatureMap();
features.put("rule", "Object_has_property_with_value_value_CHUNK");
try {
    outputAS.add(keyPhraseAS.firstNode().getOffset(),
        keyPhraseAS.lastNode().getOffset(),
        "CLIE-PropertyValue", features);
    outputAS.add(parsedAS.firstNode().getOffset(),
        parsedAS.lastNode().getOffset(),
        "CLIE-SentenceChunked", Factory.newFeatureMap());
} catch (Exception ioe) {
    ioe.printStackTrace();
}
}

```

A.4.2 np.jape

This file detects noun chunks as defined for CLIE.

```

/*
 * np.jape
 *
 * Copyright (c) 1998-2005, The University of Sheffield.
 *

```

*/

Phase: NP
 Input: Token
 Options: control = appelt

Macro: NP

```
(
  ({Token.category == CD})*
  (({Token.category == RB}|{Token.category == VBG})*
    ({Token.category == JJ, Token.kind == word})*)*
    ({Token.category == NN, Token.kind == word}
    |
    {Token.category == NNS, Token.kind == word}
    |
    {Token.category == NNP, Token.kind == word}
    |
    {Token.category == NNPS, Token.kind == word}
    |
    {Token.category == VBG} |
    ({Token.category == NNP}|{Token.category == NN})
    {Token.kind == number}(({Token.string=="."})?
    {Token.kind == number})*
  )
)
```

Macro: NPDouble

/* two nouns together - warning this may overgenerate! */

```
(
  ({Token.category == CD})*
  (({Token.category == RB}|{Token.category == VBG})*
    ({Token.category == JJ, Token.kind == word})*)*

    ({Token.category == NN, Token.kind == word}
    |
    {Token.category == NNS, Token.kind == word}
    |
    {Token.category == NNP, Token.kind == word}
    |
    )
```



```

        {Token.category == NNPS, Token.kind == word}
      )

      ({Token.category == NN, Token.kind == word}
      |
      {Token.category == NNS, Token.kind == word}
      |
      {Token.category == NNP, Token.kind == word}
      |
      {Token.category == NNPS, Token.kind == word}
      )

    )

```

Macro: NPLONG

```

/* note to remove the
 * doubleNPs being identified,
 * just replace (NP | NPDouble) with (NP)
 */

```

```

(
    (NP | NPDouble)

    ({Token.category == IN}
    (NP | NPDouble)
    ) *
)

```

Rule: NP1

Priority: 50

```

(
    (NP|NPDouble)+
):np
-->
:np.NP= {rule = "NP1"}

```

A.4.3 quotes.jape

This recognises quoted strings and generates the annotation QS.

```

/*
 * np.jape
 *
 * Copyright (c) 1998-2005, The University of Sheffield.
 *
 */

Phase: NP
Input: Token SpaceToken
Options: control = first

Rule: single_quote
({Token.string == "'" } (({Token}|{SpaceToken})+):quoted {Token.
  string == "'" })
-->
{
  String quoted = "";
  AnnotationSet quotedAS = (AnnotationSet)bindings.get("quoted");
  try {
    quoted = doc.getContent().getContent(quotedAS.firstNode().
      getOffset(),
        quotedAS.lastNode().getOffset()).toString();
    FeatureMap features = Factory.newFeatureMap();
    features.put("string", quoted);
    features.put("rule", "single_quote");
    outputAS.add(quotedAS.firstNode().getOffset(),
      quotedAS.lastNode().getOffset(),
        "QS", features);
  }
  catch(Exception ioe){

    ioe.printStackTrace();
  }
}

Rule: double_quote
({Token.string == "\"" } (({Token}|{SpaceToken})+):quoted {Token.
  string == "\"" })

```

```
-->
{
  String quoted = "";
  AnnotationSet quotedAS = (AnnotationSet)bindings.get("quoted");
  try {
    quoted = doc.getContent().getContent(quotedAS.firstNode().
      getOffset(),
        quotedAS.lastNode().getOffset()).toString();
    FeatureMap features = Factory.newFeatureMap();
    features.put("string", quoted);
    features.put("rule", "double_quote");
    outputAS.add(quotedAS.firstNode().getOffset(),
      quotedAS.lastNode().getOffset(),
      "QS", features);
  }
  catch(Exception ioe){

    ioe.printStackTrace();
  }
}
```