Alignment Based Learning for Java – A developers guide*

Stephan Schwiebert

May 15, 2009

Abstract

ABL4J is a Java port of Menno van Zaanen's Alignment Based Learning (ABL) algorithm[1]. The idea of ABL4J was a) to make ABL available for Java developers and b) to offer them a flexible alignment framework that can be extended in several ways. As ABL is a popular alignment tool (if one can say that linguistic alignment tools can be popular at all), I decided to use the codebase of ABL 1.1 for the base functionality of ABL4J. This allows developers to compare their alignment methods with ABL's methods, so one can easily see if a new algorithm is an improvement or not.

This paper is structured as follows. Section 2 will explain how ABL4J can be used within another Java application. In section 3, the architecture of ABL4J will be presented, thus demonstrating how new algorithms can be added to the framework. Section 4 will give an overview of both existing options of ABL and new options introduced with ABL4J. Finally, section 6 will explain how JUnit can be used with ABL4J to compare the processing results of ABL and its Java port.

1 Introduction

This paper is a developer's documentation to get familiar with the architecture of ABL4J. Thus, it will not explain the theoretical concepts of ABL, or the concrete implementations of ABL's algorithms. Instead, it is highly recommended to read van Zaanen (2001) and van Zaanen (2003) for a description of alignment in computational linguistics and

*Early draft version for ABL4J 0.97

[1]see http://ilk.uvt.nl/ menno/research/software/abl

Geertzen (2006) for a description of the architecture and parameters of ABL first.

As the Eclipse IDE[2] was used to develop ABL4J, the project can be checked out from its SVN repository directly to an Eclipse workspace. The project reexports all dependencies for both java applications and Eclipse plugins, thus, it can simply be added to the project dependencies of another project.

2 Usage

To use ABL4J within a Java application, only a few lines of code are needed. Assuming you added all required libraries to your project, this section shows how align, cluster, and select are executed.

```
// Initialize ABL4J: ABLInitializer initializer = new
    ABLInitializer();
Properties props = new Properties();
props.put(AblProperties.INPUT_FILE, "testdata/input/input.txt");
props.put(AblProperties.OUTPUT_FILE, "testdata/output.txt");
initializer.initialize(null, props);
PropertiesMap properties = initializer.getProperties();

// Create a new Treebank
ITreeBank treeBank = DataFactory.newTreeBank();
AblReader reader = IOFactory.getReader(properties);
reader.readTreebank(treeBank);
```

Listing 1: Programmatically configuring ABL4J

As Listing 1 shows, the *PropertiesMap* is a very important part of ABL4J, as it defines which data is processed, and how the processing within align, cluster or select is further specified. Here, most options are defined non-programmatically to keep the example simple. However, section 4 gives a detailed overview of all build-in configuration options and section 3 shows how custom algorithms can be implemented and added to the framework. Once ABL4J has been set up, a treebank can be created and filled with data to be analyzed, as shown in Listing 2. Note that the treebank returned from the factory is an interface – this allows to use different treebank implementations without modifying any code.

The reader which is used to read a treebank is also returned by a factory method, thus one can implement a custom reader for any new treebank format and simply configure the framework to use this reader.

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

```
// Create a new Treebank
ITreeBank treeBank = DataFactory.newTreeBank();
AblReader reader = IOFactory.getReader(properties);
reader.readTreebank(treeBank);
```

Listing 2: Creating a new Treebank

For example, ABL4J comes with a treebank reader for the Tiger Corpus[3] format, which converts the manually created annotations within that corpus to ABL compatible data structures. A treebank reader for the British National Corpus will be added soon.

To process the data loaded into the treebank, instances of Align, Cluster and Select have to be created, configured and executed. Listing 3 show how this can be done.

```
// Create, configure and execute Align, Cluster, and Select
    components
Align align = new Align();
Cluster cluster = new Cluster();
Select select = new Select();
align.configure(properties);
cluster.configure(properties);
select.configure(properties);

// Execute components
align.execute(treeBank);
cluster.execute(treeBank);
select.execute(treeBank);
```

Listing 3: Running ABL Components

Finally, the processed data should be stored somewhere, so in listing 4 the IOFactory is asked for a writer to store the treebank.

```
// Write treebank
AblWriter writer = IOFactory.getWriter(properties);
writer.writeTreebank(treeBank);
```

Listing 4: Storing the new Treebank

Note that in listing 2 and 4 no file names are used – this information is contained within the PropertiesMap given to the factory. This might be a bit confusing, as the IO-code does not show which files are

[3]see http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERCorpus/ for details

processed, but on the other hand one can freely connect ABL4J to any storage system, in which, for example, custom readers and writers use custom properties to load and store treebanks within a database, use data already contained in memory, or something similar.

The complete listing can be found in class `Example1`. If executed, some Log4J-Messages should be printed and the results of the alignment should be found in file *testdata/output.txt*.

3 Customization

Most parts of ABL4J make intensive use of Java interfaces and thus offer several ways to add new functionality. In general, all these interfaces are related to one of the following:

- Internal data structures, like the implementation of ITreebank or IConstituent,
- IO-operations to serialize and deserialize treebanks, or
- Algorithms that perform the align, cluster, or select step of ABL.

The rest of this section will give a basic overview of these parts of ABL4J.

3.1 Data Structures

If the implemented data structures do not fit the requirements of a new algorithm, or if the architecture of an application that uses ABL4J needs a different implementation, the interfaces found in package `org.schwiebert.abl4j.data` can be reimplemented. Figure 1 shows a class diagram of these interfaces.

To make use of the new classes, the corresponding properties (see table 4.2) must be set to the fully qualified class name of the custom classes. Additionally, the mapping of terminal symbols can be modified. ABL originally maps words (strings) to integers and makes a reference to these in its data structures, however, if another kind of data shall be aligned, a new `IWordMapping` can be implemented. In most cases, only a new `java.util.Comparator` has to be passed to the constructor of the default implementation `WordMapping`, as shown in Listing 5 for the default word mapping implemented for strings. As with data structures, the word mapping must be set by the corresponding property (see table 4.2).

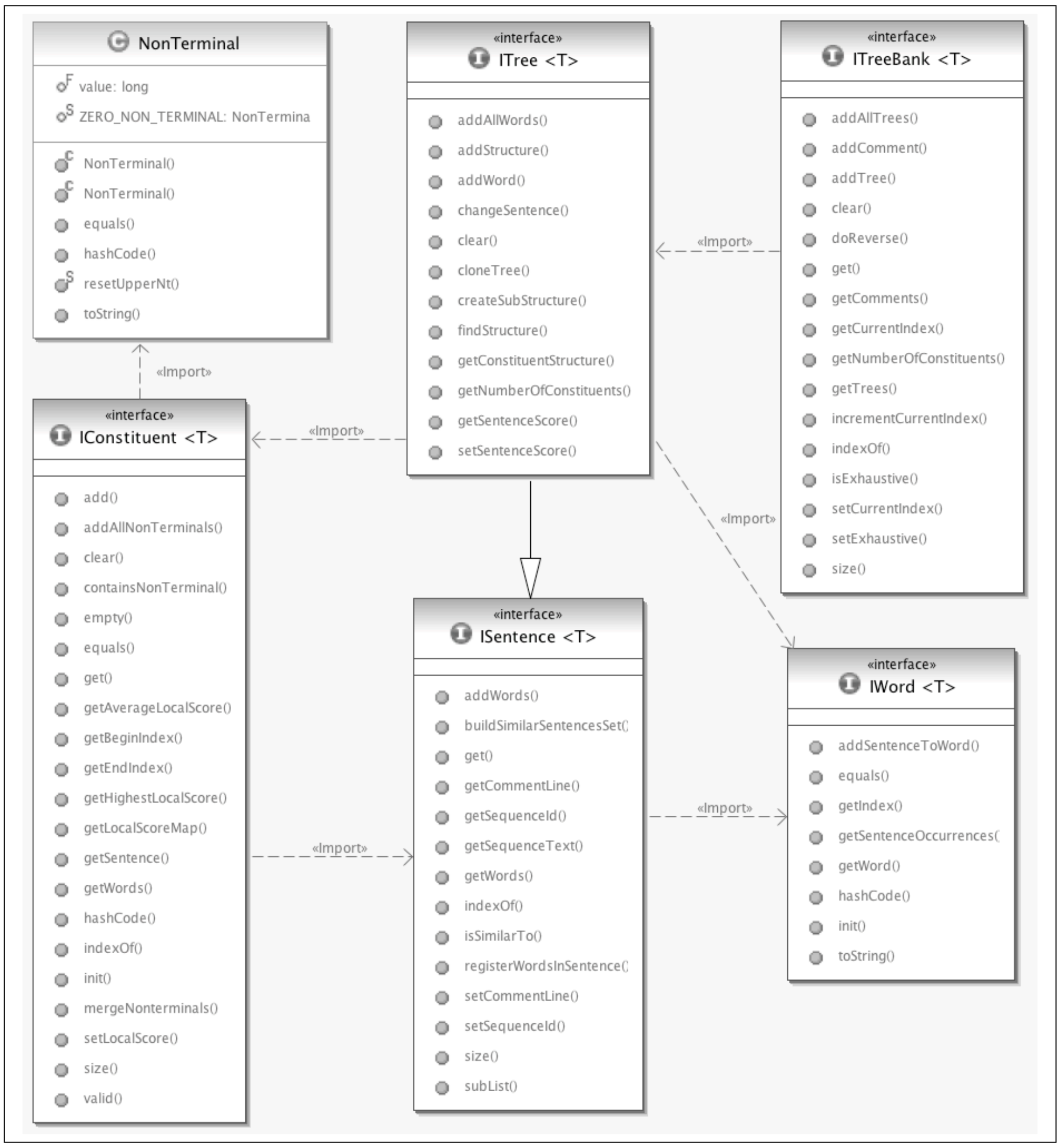

Figure 1: Internal Data Structures

```
public StringMapping() {
        super(new Comparator<String>() {
                public int compare(final String o1, final String o2) {
                        return o1.compareTo(o2);
                }
        });
}
```

Listing 5: Constructor of class StringMapping

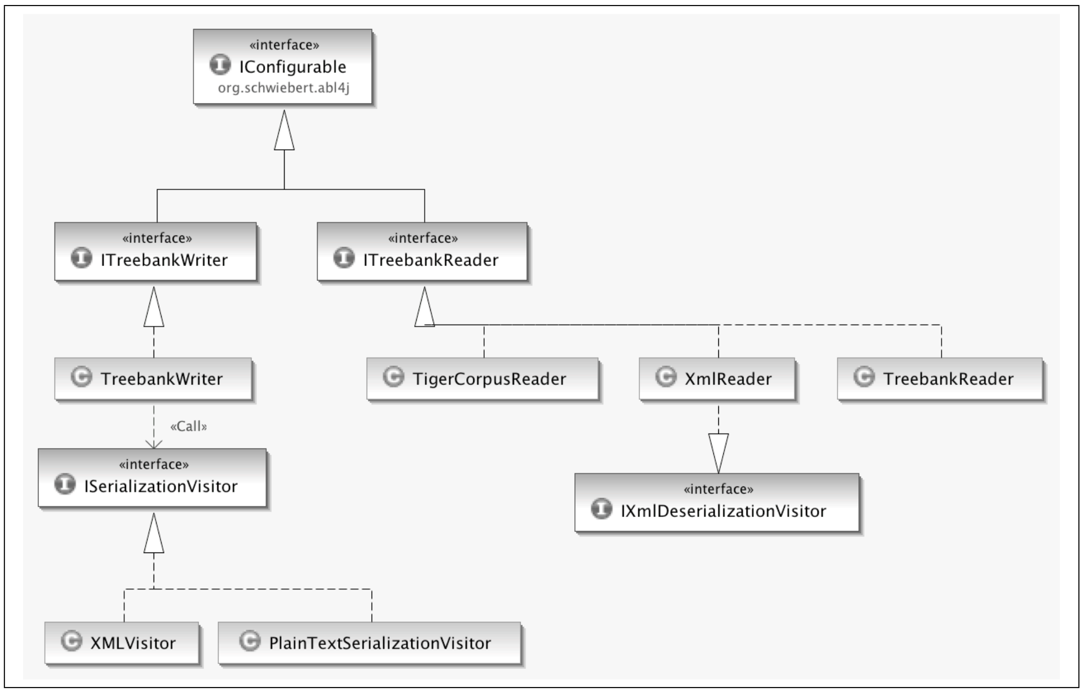

Figure 2: IO-System in ABL4J

3.2 IO

As listings 2 and 4 showed, IO-operations are encapsuled in `ITreebankReader` and `ITreebankWriter`. To support new data formats, one can either implement these interfaces (or one of them to support a custom format as input or output only, as in case of the `TigerCorpusReader`), or extend one of the existing implementations. Figure 2 gives an overview of the IO-related classes and interfaces which are currently available in ABL4J.

Additionally, a *visitor*-pattern has been integrated in ABL4J, so that serialization and deserialization can be further simplified: If, for instance, a new data format should be written, the `ISerializationVisitor`-interface can be reimplemented and set for use within the default serializer via the `serialization.visitor` property (see section 4.2). Thus, no code for visiting the treebank data structure must be implemented. However, as the visitor pattern requires that the structure of the visited data is known, a deserialization visitor is available only for deserializing XML formats.

Note that ABL4J does not define the kind of data source or destination – instead, readers and writers can be implemented to use files, streams, or something similar. The existing readers and writers support files and streams by using the properties `input.file` or `input.stream` for reading and `output.file` or `output.stream` for writing. If none of these properties is set, the default streams `System.in` and `System.out` are used. In this case, the streams won't be closed after serialization or deserialization.

3.3 Algorithms

Customization of data structures and IO functionality is probably required only to integrate ABL4J into another architecture. To modify the produced structures, or to test new hypotheses, new algorithms can be added to ABL4J by implementing one of the interfaces `AlignmentMethod`, `ClusterMethod`, or `SelectMethod`. Figure 3 gives an overview of the existing alignment methods of ABL, and how they are integrated into ABL4J.

In general, an alignment method compares tree structures and generates constituent hypotheses. A clustering method groups these hypotheses, and a select method detects and removes incompatibilities between hypotheses – see van Zaanen (2003) for a detailed description of these ideas.

4 Configuration

The original ABL programs *align*, *cluster* and *select* can be configured by several command line options. Most of these options can be used to launch the ABL4J ports of these programs, however, some of them are not supported (as, for instance, ABL4J uses Log4J for logging, so no custom log architecture was implemented). Also, if ABL4J is used within another Java application, other ways of configuring parameters are needed. This section describes the way ABL4J can be configured.

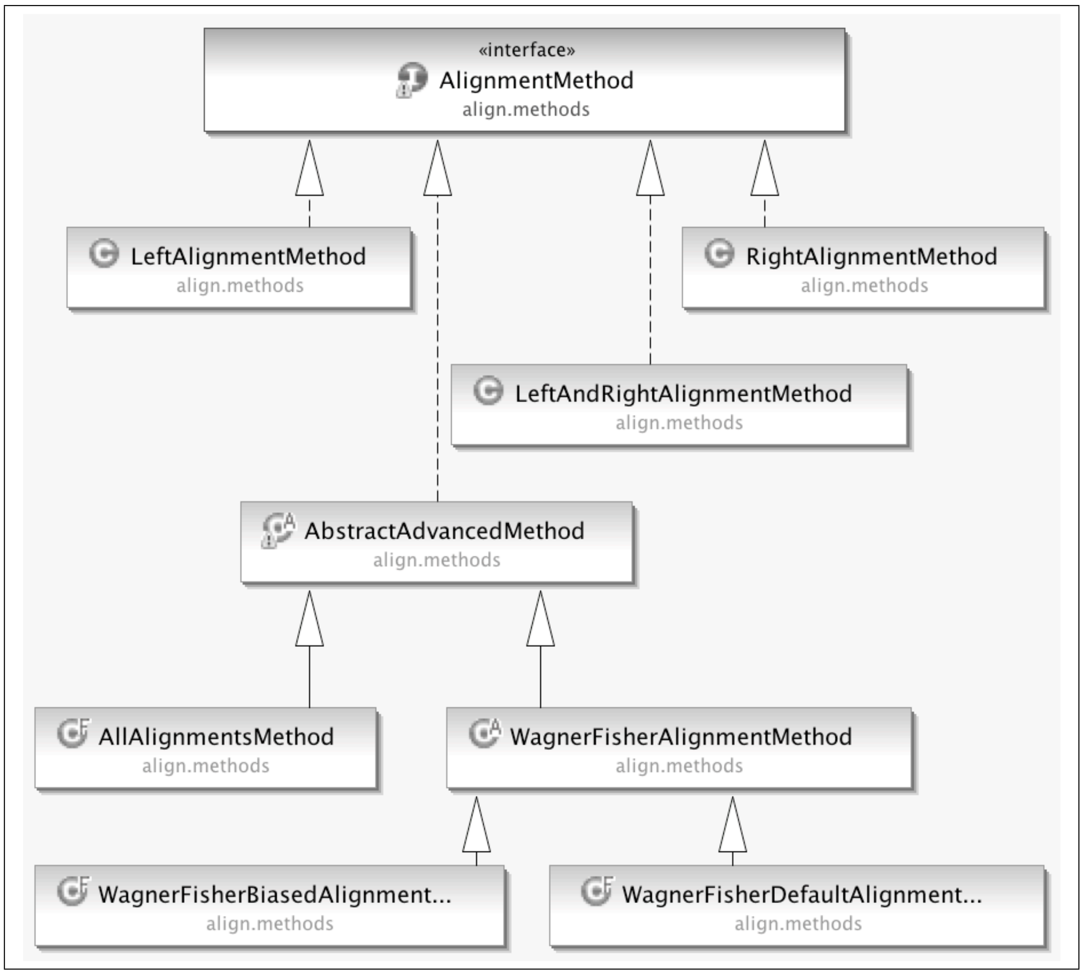

Figure 3: Hierarchy of implemented alignment methods

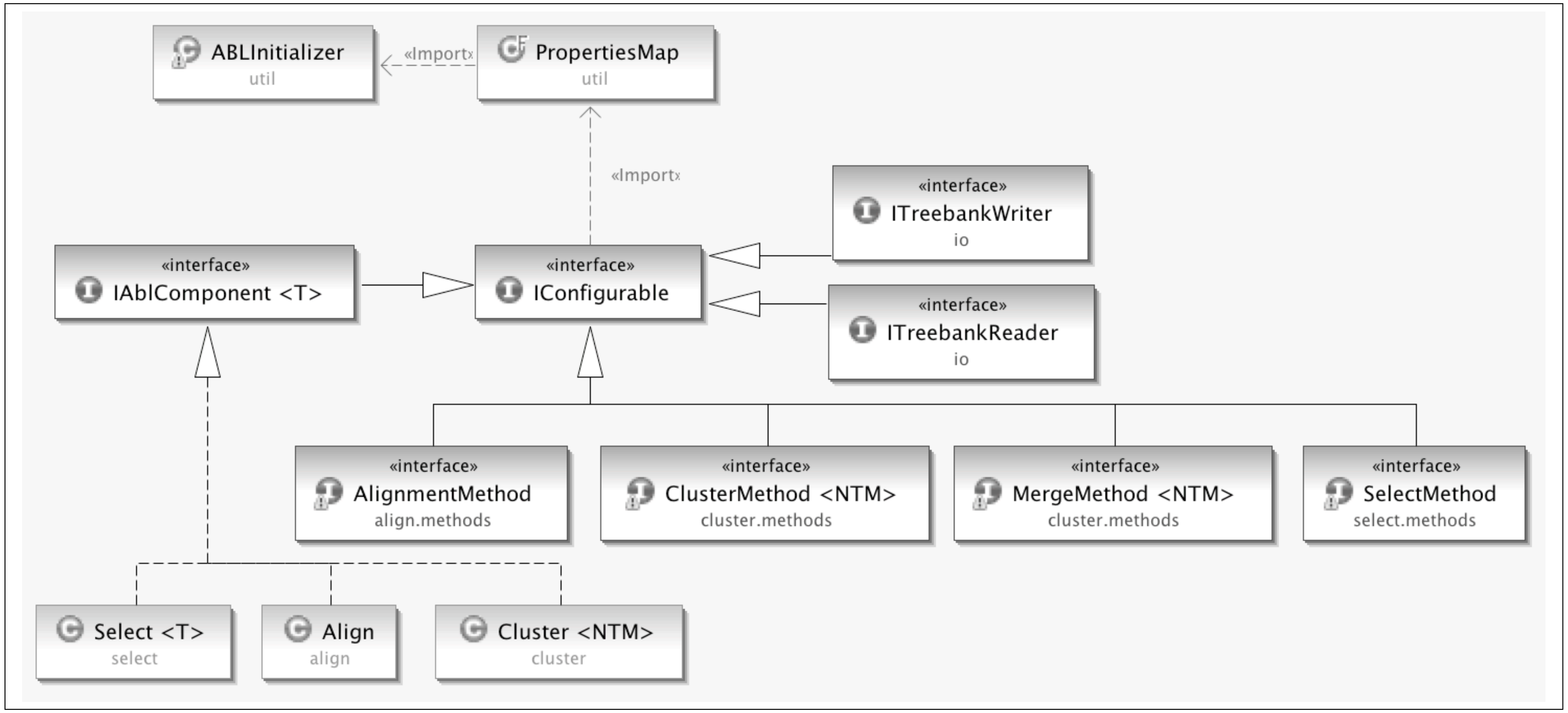

Figure 4: Overview of configurable elements in ABL4J

4.1 Configuration sequence

Before one of the classes `Align`, `Cluster`, or `Select` can be used, it must first be configured by calling `configure(PropertiesMap map)`. This method starts the configuration process, which is based on Java's properties system. First, some properties will be set to default values to simplify the required configuration – see tables 4.1 and 4.2 for details. Than, ABL4J will search for a file called *abl4j.properties* on the classpath. If found, the properties defined there will be added to the set of properties, probably overriding any default values set before. Finally, if ABL4J is launched from a command line, any options defined there will be added to the properties map (again overriding already defined properties with the same name). If ABL4 is integrated in another java application, this step will be skipped – instead, properties can be set programmatically by calling `PropertiesMap.put(String key, String value)` before an ABL4J component is initialized (see section 2).

Name	ABL option	ABL4J property	Supported values	ABL4J values	Default value	Component
Input File	-i	input.file	any filename	any filename	null (System.in)	all
Output File	-o	output.file	any filename	any filename	null (System.out)	all
Align type	-a	align.method	a, l, r, b, aa, wm, wb (but not st1…st4)	fully qualified class name	…align.methods. AllAlignment	align
Part type	-p	part.type	e, u, b	enum PartType	unequal	align
Exclude Empty	-e	align.exclude.empty	–	boolean	false	align
No Merge	-n	align.nomerge	–	boolean	false	align
Exhaustive	-x	align.exhaustive	–	boolean	false	align
Select Type	-s	select.type	f, t, c, b	enum Select-Type	…select.methods. ConstituentSelect Method	select
Seed	??	seed	integer	integer	0	align, select
Verbose	-v	not supported	–	–	false	all
Preserve	-m	select.preserve	–	boolean	false	select

Table 1: ABL options and their ABL4J equivalents. Note: All ABL4J properties and fully qualified class names begin with the prefix *org.schwiebert.abl4j*.

As ABL4J uses Log4J for logging, ABLs logging flags are not supported. Instead, logging can be configured by modifying file *log4j.properties* found in the project's root directory[4].

4.2 ABL4J properties

ABL4J adds several options to the set of options used by ABL. Most of these are architecturally required and cannot be defined via command line, as they are intented to be used only in applications that use ABL4J as a library. Table 4.2 gives an overview of these options. The options *compatibility.mode* is used for debugging, in particular to compare the results of ABL and ABL4J. This is the only option that can be used from command line by adding the parameter *-j*. See section 6 for further details. As parts of ABL4J support multithreading, the number of threads can be defined by setting *threads* to an integer greater than 1 – see section 5. If *reset.upper.nt* is set to `true`, during initialization of align, cluster, or select, the largest unused non terminal id is set to 0. The options *cluster.method* and *merge.method* can be used to partially modify the clustering phase of ABL instead of implementing a new cluster method.

[4]If you are not familiar with Log4J, visit http://logging.apache.org/log4j/.

Name	Property	Supported values	Default value	Component
Compatibility Mode	compatibility.mode	boolean	false	all
Reader Class	reader.class	fully qualified class name	. . . io.TreebankReader	all
Writer Class	writer.class	fully qualified class name	. . . io.TreebankWriter	all
Serialization Visitor	serialization.visitor	fully qualified class name	. . . io. PlainTextSerializationVisitor	all
Cluster Method	cluster.method	fully qualified class name	. . . cluster.methods. ABLClusterMethod	cluster
Merge Method	merge.method	fully qualified class name	. . . cluster.method. ABLClusterMethod	cluster
Input Encoding	input.encoding	String	UTF-8	all
Output Encoding	output.encoding	String	UTF-8	all
Number of Threads	threads	integer	1	align
Comparism Mode	comparism.mode	boolean	false	all
Constituent Class	constituent	fully qualified class name	. . . data.Constituent	all
Sentence Class	sentence	fully qualified class name	. . . data.Sentence	all
Treebank Class	treebank	fully qualified class name	. . . data.Treebank	all
Word Class	word	fully qualified class name	. . . data.Word	all
Tree Class	tree	fully qualified class name	. . . data.Tree	all
Reset Upper NT	reset.upper.nt	boolean	true	all
Word Mapping	word.mapping	fully qualified class name	. . . util.StringMapping	all
Input Stream	input.stream	instance of java.io.InputStream	null	all
Output Stream	output.stream	instance of java.io.OutputStream	null	all

Table 2: Additional ABL4J properties. Note: All ABL4J properties and fully qualified class names begin with the prefix *org.schwiebert.abl4j*.

5 Multithreading

In general, it is possible to execute multiple instances of ABL4J within the same virtual machine, even with different parameters. However, as this feature has not been tested well yet, it is still experimental and might cause problems.
Besides that, the align and select phases of a single instance of ABL4J can be run in several threads, thus improving the overall performance on multi core or multi processor pc's. However, to support this feature, it is important that new algorithms or data structures do not use *static* fields, as this would probably lead to unexpected results. Instead use, for instance, `java.lang.InheritableThreadLocal` to create "'pseudo-static"' variables.

6 JUnit – comparing ABL and ABL4J

While porting ABL to Java, a JUnit Test Suite was used to compare the treebanks produced by ABL and ABL4J. The goal was to produce exactly the same structures that were produced by ABL, as this is what one expects if using a ported framework. However, there were a few differences between Java and C++ that required modifications of both ABL4J and ABL. So, if you want to ensure that ABL4J does work as expected, you will unfortunately have to do some work first. Otherwise you can skip this section.

ABL4J can be used with a *compatibility flag* that forces the program to produce exactly the same output as the C++ version. For instance, if it is set to `true`, all random numbers are generated by calling a native C function, so random based parts (like the *both* alignment method or the select component) execute equally. However, another difference between C and Java is related to rounding double-values, and as a workaround the related functions of ABL were modified. So, in order to compare the data produced by ABL and ABL4J, a few changes must be made to the ABL source code, as shown in 6 and 7.

Finally, the native random code used by ABL4J in compatibility mode must be compiled – the source and header files can be found in dirctory *c_src* of ABL4J. The compiled library than must be added to the java.library.path, which can be done by placing it directly in the project root directory.
Note that all these modifications are optional, as long as you do not expect that ABL4J will produce *exactly* the same results as ABL. However, if you want to compare both versions and successfully compiled all required C++-code, you can place the generated executables of align, cluster and select in the directory *orig* of the ABL4J project

```
Replace

Sub_dis(Ran b1, Ran e1, Ran b2, Ran e2) throw()
     :Sub<Ran>(b1, e1, b2, e2) {
      for(; b1!=e1;b1++) { len1++; }
      for(; b2!=e2;b2++) { len2++; }
   };

with

Sub_dis(Ran b1, Ran e1, Ran b2, Ran e2) throw()
     :Sub<Ran>(b1, e1, b2, e2) {
        len1 = 0; len2 = 0;
      for(; b1!=e1;b1++) { len1++; }
      for(; b2!=e2;b2++) { len2++; }
   };
```

Listing 6: Required Changes in Align.cpp

and then run the JUnit tests (in correct order: align, cluster, select) found in the package org.schwiebert.abl4j.tests. These tests will use the content of file *testdata/input/input.txt*, call both C++ and Java versions of the program with every parameter variant and compare the processing results. They will fail if the results differ.

7 Licence

ABL4J is licensed unter the GNU Lesser General Public License (LGPL). Thus, it can be used in any other software project without further license requirements. However, if you add new features to it (as, for instance, new algorithms or support of new data formats), it would be great if you would allow me to add them to the library, so that other developers can benefit from them.

```
Insert the following two functions

bool isLess(double a, double b) {
        double result = a - b;
        result += 0.000001;
        return result < 0;
}

bool isLessOrEqual(double a, double b) {
        double result = a - b;
        if (result + 0.000001 < 0) {
                return true;
        }
        if(result < 0.000001) {
                return true;
        }
        return false;
}

and replace the tests

 ...
 if (((new_p=compute_combined_probability(t, res, prob))<=best_p) ||(uninit)) {
 ... if ((new_p>=0)&&(new_p<best_p)) {
 ...
     }
 }

 in select_prob_in_range with

...
 new_p = compute_combined_probability(t, res, prob);
 if (isLessOrEqual(new_p, best_p) ||(uninit)) {
 ...
        if ((new_p>=0)&&(isLess(new_p, best_p))) {
        ...
        }
        ...
 }
```

Listing 7: Required Changes in Select.cpp

Bibliography

[Geertzen 2006] Geertzen, Jeroen: Alignment-Based Learner Reference Guide. (2006), Feb. http://ilk.uvt.nl/~menno/personal_files/software/abl/abl-1.1.tar.gz

[van Zaanen 2001] Zaanen, Menno van: Bootstrapping Syntax and Recursion using Alignment-Based Learning. In: *Arxiv preprint cs.LG* (2001), Jan. http://arxiv.org/abs/cs.LG/0104007

[van Zaanen 2003] Zaanen, Menno van: Alignment-Based Learning versus Data-Oriented Parsing. (2003), Aug. http://citeseer.ist.psu.edu/667428