Sentence Boundary Recognition in *SProUT*

Jakub Piskorski
German Research Center for Artificial Intelligence
Stuhsatzenhausweg 3, 66123 Saarbrücken, Germany
piskorsk@dfki.de

VERSION JANUARY 2005
(SPROUT VERSION 4.0 AND HIGHER)

1 Introduction

SProUT now provides a linguistic processing resource for sentence/utterance boundary recognition which can also be used separately as a subcomponent in other applications. This paper briefly describes this component, issues concerning creation of indispensable resources, and gives some code examples for programmers.

SproUT's sentence boundary recognition component follows the rule-based approach, and is based on a user-definable list of potential sentence boundary markers, a list of non-final text items (e.g. non-final abbreviations), a list of non-initial text items (e.g., some capitalized abbreviations which do not start a sentence), and a list of non-initial prefixes (e.g., symbols like '(' or '%'). For the sake of clarity, we give below an example of such resources for German language.

Potential sentence boundary markers: `.` `!` `?` `"`

Non-final text items: `Prof.` `Dr.,` `z.B.`

Non-initial text items: `AG und Co.,` `Mrd.,` `Ltd.`

Non-initial prefixes: `(` `[` `{` `<` `%`

Note that non-initial prefixes must not be necessarily followed by a whitespace, whereas non-initial text items must be followed by a whitespace. For this reason we make the distinction between these two lists.

The component applied to an input text returns an array of sentence/utterance boundary markers (character positions). When applied within SProUT, these results are not directly visible, but the set of the grammar rules can be split into rules which are sentence-boundary sensitive and sentence-boundary insensitive, i.e., sentence-boundary sensitive rules can not match a text fragment which contains a sentence/utterance boundary.

Furthermore, the results returned by the sentence/utterance boundary recognizer may be fine-tuned at a later stage via comparison with text spans matched by another component, and through exclusion of all sentence/utterance boundaries which appear in any of these text spans. For instance, named-entities often include non-trailing periods (e.g., *Prof. Dr. hab. John Smith*) In this way, the user has the possibility of improving the results since many of the ambiguities concerning the function of punctuation signs can be resolved at higher-level of the linguistic analysis. Additionally, a so called *two-end-of-line* rule can be used on demand (default option) in order to treat two or more consecutive end-of-line symbols as an evidence for sentence/utterance boundary.

2 Preparing and Compiling Sentence Boundary Recognizer Resources

The resources needed for compiling a sentence boundary recognizer consist of four files containing respectively potential sentence boundary markers, non-final text items, non-initial text items, and non-initial prefixes. Note that each line of these files contains a single entry.

When these four files are provided, we can turn to the compilation process. For running the sentence boundary recognizer compiler, a configuration file is needed, which defines the following properties:

- a path to the file containing the list of potential boundary markers
- a path to the file containing the list of non-final items
- a path to the file containing the list of non-initial items
- Path to the file containing the list of non-initial prefixes
- a name of the character set used for decoding input files (e.g., US-ASCII, ISO-8859-1, UTF-8, UTF-16BE). Please refer for further details to `java.nio.charset.Charset` in the online JAVA API documentation available at `http://www.java.sun.com`

- a path to the output file containing the binary compressed representation of the resources for the sentence boundary recognizer (by a convention compiled sentence boundary resources should have an extension `ssb`)

The example given below illustrates how a configuration file should look like.

```
BoundaryMarkersFile=BoundaryMarkers.txt
NonFinalItemsFile=NonFinalItems.txt
NonInitialItemsFile= NonInitialItems.txt
NonInitialPrefixFile= NonInitialPrefix.txt
CharacterSet=ISO-8859-1
OutputFile=SentenceBoundary.ssb
```

When the definition of the configuration file is completed we can finally compile the resources by running the following script[1]:

MS Windows version:

```
java -Xmx640M -cp %1
de.dfki.lt.sprout.runtime.sentenceBoundary.CompilerForSentenceBoundaryRecognizer %2
```

Linux version:

```
export LD_LIBRARY_PATH=.  // specifies the path where dynamic link libraries are stored
java -Xmx640M -cp $1
de.dfki.lt.sprout.runtime.sentenceBoundary.CompilerForSentenceBoundaryRecognizer $2
```

where %1 ($1) specifies the path to: (a) the directory containing *SProUT* Java archive (`sprout.jar` or `sprout.zip`) or a path to the root directory containing the compiled `CompilerForSentenceBoundaryRecognizer` class, and (b) the `log4j-1.2.8.jar` archive[2] (used for logging since version 3.7). The second parameter %2 ($2) is a path to the configuration file described earlier. Any messages concerning errors encountered during the compilation are displayed via standard output. Since the regular compiler is invoked in the compilation process, make sure that the corresponding dll-library (`RC_JNI.dll` and `libRC_JNI.so` libraries for MS Windows and Linux respectively) is visible by your operating system. In the MS Windows environment it is sufficient to store this library in the working directory or to make the library visible via adding the corresponding absolute path of the library to the environment variable `PATH`. In case of Linux, an additional initial line containing the export directive has to be added to the script which defines the environment

[1] Please note that currently compiling sentence boundary recognizer resources is only possible under Linux or MS Windows since the dynamic library providing the functionality of the regular compiler that is used in the compilation process, is not available for other operating systems.

[2] For internal DFKI staff: The latest versions of all jar archives and external libraries which are used by *SProUT* can be found in DFKIs internal directory /project/cl/sprout/test/resource/lib. External users will find all these libraries in their main *SProUT* directory.

variable `LD_LIBRARY_PATH` ant sets its value to the path where the library is stored. In our script example above, this variable is set to the current directory. Do not forget to modify it if necessary.

Further, the directory in which you compile your resources must be writable, since the compiler produces some intermediate temporary files. It is recommended that users do not use any of the internal *SProUT* directories for compiling their resources.

3 Running the Sentence Boundary Recognizer

When using the sentence/utterance boundary recognizer within the SProUT grammar development environment, you have to choose the option *Menu > Tools > Configure* components which leads you to the configuration window in which you can launch an instance of such component in the similar way to installing other processing resources. The only two fields which have to be specified are the *resource file* field (compiled resources for sentence boundary recognizer), and *Two-end-of-line rule* activation field.

If you wish to use the sentence boundary recognizer independently and test it outside of *SProUT* on any textual data, the following script can be used:

```
java -Xmx640M -cp <LibPath>
de.dfki.lt.sprout.runtime.sentenceBoundary.ApplySentenceBoundaryRecognizer
     [-t] <SentenceBoundaryRecognizerFile> <TokenizerConfigFile> <InputFile> <OutputFile>

        <LibPath> specifies: (a) the path to the directory containing the SProUT Java archive
                  or a root directory containing the compiled ApplySentenceBoundaryrecognizer
                  class, and (b) the path to the log4j-1.2.8.jar archive

        <-t> a switch for deactivating TwoEndOfLineRule

        <SentenceBoundaryRecognizerFile> a path to the file containing the resources


        <TokenizerConfig> a path to the file containing the tokenizer configuration

        <InputFile> a path to the file containg the input text

        <OutputFile> a path to the output file
```

Please note that this script automatically runs SProUTs tokenizer as a subroutine. However the user has the possibility to use any tokenizer he wants with SproUTs sentence boundary recognizer. How to proceed in such a case will be explained later a sample code.

An example: In order to apply the sentence boundary recognizer with *two-end-of-line* rule activated to the file **`sample.txt`**, whereas the resources are stored in the file **`SentenceBoundary.ssb`**, the tokenizer configuration is saved in the file **`TokConf.txt`**, and the output should be written to the file **`result.txt`**, following script must be used:

```
java -Xmx640M -cp sprout.jar;log4j-1.2.8.jar
de.dfki.lt.sprout.runtime.sentenceBoundary.ApplySentenceBoundaryRecognizer
SentenceBoundary.ssb TokConf.txt sample.txt result.txt
```

Developers who would like to perform the same action in their own applications would need to generate the following piece of code (in a simplified form):

```
import java.io.*;
import java.util.*;
import de.dfki.lt.sprout.runtime.tokenizer.*;
import de.dfki.lt.sprout.runtime.gazetteer.*;
import de.dfki.lt.sprout.runtime.sentenceBoundary.*;

// Initialize the sentence boundary recognizer
SentenceBoundaryRecognizer E = new SentenceBoundaryRecognizer();

// Read the compiled resources from file
E.readFromFile("SentenceBoundary.ssb");

// Activate the two-end-of-line rule
E.activateTwoEndOfLineRule();

// Read the input text from a file into a string

byte[] byteBuffer = null;
try
  { File in_name = new File("sample.txt");
    byteBuffer = new byte[(int)in_name.length()];
    FileInputStream in = new FileInputStream("sample.txt");
    in.read(byteBuffer);
    in.close();
  } catch(Exception e)
     { System.out.println("IO Exception"); System.exit(0); }

String INPUT = new String(byteBuffer);

// Run the SProUT tokenizer
// If the user wants to use other tokenization software this part of code has to be
// modified\exchanged appropriately

TokenIterator pTI = null;
pTI = RunTokenizer.RunTok("sample.txt","TokConf.txt");
pTI.reset();

// Apply the sentence boundary recognizer

// Allocate an array for storing the results returned by the sentence boundary recognizer
int[] Boundaries = null;

// Since sentence boundary recognizer takes as input an array of 'sbToken' objects,
// tokenization information must be post-processed in order to create such an array

// For converting SProUTs tokens one can use the following method for creating
// an array of 'sbToken' objects
sbToken[] Tokens = SentenceBoundaryRecognizer.ConvertSproutTokens(pTI);

// If You are using different tokenizer, do perform the conversion of
// tokenization information into 'sbToken's here and replace the above call to
// the ConvertSproutTokens(pTI) to something else
// More details on the class 'sbToken' can be found in the Appendix A

// Run the senetence boundary recognizer
Boundaries = E.ComputeBoundaries(INPUT.toCharArray(),Tokens);
```

For further details, please refer to the java code documentation of the corresponding *SProUT* package in Appendix A.

APPENDIX A – JAVA DOCUMENTATION

JAVA DOCUMENTATION FOR THE CLASS AbstractSentenceBoundaryRecognizer

de.dfki.lt.sprout.runtime.sentenceBoundary

Class AbstractSentenceBoundaryRecognizer

java.lang.Object
  └ **de.dfki.lt.sprout.runtime.sentenceBoundary.AbstractSentenceBoundaryRecognizer**

Direct Known Subclasses:

SentenceBoundaryRecognizer

---

public abstract class **AbstractSentenceBoundaryRecognizer**
extends java.lang.Object

General abstract class for sentence/utterance boundary recognition component.

Constructor Summary

AbstractSentenceBoundaryRecognizer()

Method Summary

abstract int[]	**ComputeBoundaries**(char[] InputText, sbToken[] Tokens) Returns an array of indices of sentence boundaries.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AbstractSentenceBoundaryRecognizer

public **AbstractSentenceBoundaryRecognizer**()

Method Detail

ComputeBoundaries

public abstract int[] **ComputeBoundaries**(char[] InputText,
sbToken[] Tokens)

Returns an array of indices of sentence boundaries.

Parameters:

`InputText` - array containing the input text.

`Tokens` - an array of tokens (@see de.dfki.lt.runtime.sentenceBoundary.sbToken.)

Returns:

an array of integers indicating indices of character positions which represent sentence/utterance boundaries.

JAVA DOCUMENTATION FOR THE CLASS SentenceBoundaryRecognizer

---

de.dfki.lt.sprout.runtime.sentenceBoundary

Class SentenceBoundaryRecognizer

java.lang.Object
└ de.dfki.lt.sprout.runtime.sentenceBoundary.AbstractSentenceBoundaryRecognizer
└ **de.dfki.lt.sprout.runtime.sentenceBoundary.SentenceBoundaryRecognizer**

---

public class **SentenceBoundaryRecognizer**
extends AbstractSentenceBoundaryRecognizer

This class implements sentence/utterance boundary recognition component. It is based on a user-definable list of potential sentence boundary markers, a list of non-final text items (e.g. abbreviations), a list of non-initial text items (e.g., some capitalized abbreviations which do not start a sentence), and a list of non-initial prefixes (e.g., symbols like '(' or '%'). Note that non-initial prefixes must not be necessarily followed by a whitespace, whereas non-initial text items must be followed by a whitespace (for this reason we make the distinction between the two lists).

The component applied to an input text returns an array of sentence/utterance boundary markers (character positions). Further, the results can be fine-tuned via comparison with text spans matched by another component (e.g., named-entities often include non-trailing periods) through exclusion of all sentence/utterance

boundaries which appear in any of these text spans. Additionally, a so called two-end-of-line rule can be used on demand (default option) in order to treat two or more consecutive end-of-line symbols as an evidence for sentence/utterance boundary.

Constructor Summary

SentenceBoundaryRecognizer()	A standard constructor for creating objects of type SentenceBoundaryRecognizer

Method Summary

void	**activateTwoEndOfLineRule**() This method allows for activating the two-end-of-line rule.
boolean	**compileResources**(java.lang.String BoundaryMarkersFile, java.lang.String NonFinalItemsFile, java.lang.String NonInitialItemsFile, java.lang.String NonInitialPrefixFile, java.lang.String InputCharSet, java.lang.String OutputFile) Compiles the resources for Sentence Boundary Recognizer
int[]	**ComputeBoundaries**(char[] InputText, sbToken[] Tokens) Returns an array of indices of sentence boundaries.
boolean	**configureLogger**(java.lang.String LoggerPropertyFile) Configure the logger according to the Logger property file
static sbToken[]	**ConvertSproutTokens**(TokenIterator pTI) This method converts tokens generated by SProUTs' tokenizer into de.dfki.lt.runtime.sentenceBoundary.sbToken objects which are required as input to the method for computing sentence/utterance boundaries
void	**deactivateTwoEndOfLineRule**() This method allows for deactivating the two-end-of-line rule.
java.lang.String	**getCharacterSet**() Returns the Character set associated with the sentence boundary recognizer.
boolean	**isActiveTwoEndOfLineRule**() This method can be used for checking if two-end-of-line rule is active.
boolean	**isWhitespace**(char c) Checks if an input character is a whitespace.
boolean	**readFromFile**(java.lang.String filename) Reads a SentenceBoundaryRecognizer object from a binary file.
void	**readFromStream**(java.nio.ByteBuffer b) Reads a SentenceBoundaryRecognizer object from a Bytebuffer.
static int[]	**ReviseBoundaries**(int[] currentBoundaries, sbToken[] Tokens)

	This method allows for fine-tuning the results via filtering out sentence/utterance boundaries which appaer within specified text spans.
boolean	**saveToFile**(java.lang.String filename) Saves a SentenceBoundaryRecognizer object in binary file.
void	**setBasicLogger**() Activates the default basic logger (standard output)
void	**setSimpleLogger**(java.lang.String FileName) Configures the logger: Messages are redirected to the input file, and the level is set to 'INFO'.
void	**writeToStream**(java.io.DataOutputStream s) Writes a SentenceBoundaryRecognizer object in to a DataOutputStream

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SentenceBoundaryRecognizer

public **SentenceBoundaryRecognizer**()
A standard constructor for creating objects of type SentenceBoundaryRecognizer

Method Detail

activateTwoEndOfLineRule

public void **activateTwoEndOfLineRule**()
This method allows for activating the two-end-of-line rule.

---

deactivateTwoEndOfLineRule

public void **deactivateTwoEndOfLineRule**()
This method allows for deactivating the two-end-of-line rule.

---

isActiveTwoEndOfLineRule

public boolean **isActiveTwoEndOfLineRule**()
This method can be used for checking if two-end-of-line rule is active.

Returns:
`true` if the two-end-of-line rule is active, or `false` otherwise.

---

writeToStream

public void **writeToStream**(java.io.DataOutputStream s)
throws java.io.IOException
Writes a SentenceBoundaryRecognizer object in to a DataOutputStream

Parameters:
`s` - denotes the DataOutputStream to be used
Throws:
`java.io.IOException`

---

saveToFile

public boolean **saveToFile**(java.lang.String filename)
Saves a SentenceBoundaryRecognizer object in binary file. This method is not based on standard JAVA serialization, but uses specific file format.
Parameters:
`filename` - denotes the file to be used
Returns:
`true` if this operation was succesfull, or `false` otherwise

---

readFromStream

public void **readFromStream**(java.nio.ByteBuffer b)
Reads a SentenceBoundaryRecognizer object from a Bytebuffer.
Parameters:
`b` - denotes the ByteBuffer to read from

---

readFromFile

public boolean **readFromFile**(java.lang.String filename)
Reads a SentenceBoundaryRecognizer object from a binary file. This method is not based on standard JAVA serialization, but uses specific file format.
Parameters:
`filename` - denotes the file to be used
Returns:
`true` if this operation was succesfull, or `null` otherwise

---

getCharacterSet

public java.lang.String **getCharacterSet**()
Returns the Character set associated with the sentence boundary recognizer.

ReviseBoundaries

public static int[] **ReviseBoundaries**(int[] currentBoundaries,
sbToken[] Tokens)

This method allows for fine-tuning the results via filtering out sentence/utterance boundaries which appaer within specified text spans. This method is supposed to be used at a higher stage, e.g. after named-entity recognition since named entities might include some non-trailing periods.

Parameters:

`currentBoundaries` - - an array representing the current list of sentence/utterance boundaries (character positions)

`Tokens` - an array of text spans which will be used for filtering out current sentence/utterance boundaries, i.e. all boundary markers which appear in these text spans will be filtered out.

Returns:

an array representing the modified list of sentence/utterance boundaries

compileResources

public boolean **compileResources**(java.lang.String BoundaryMarkersFile,
java.lang.String NonFinalItemsFile,
java.lang.String NonInitialItemsFile,
java.lang.String NonInitialPrefixFile,
java.lang.String InputCharSet,
java.lang.String OutputFile)

Compiles the resources for Sentence Boundary Recognizer

Parameters:

`BoundaryMarkersFile` - a name of the file containing a list of potential sentence boundary markers

`NonFinalItemsFile` - a name of the file containing a list of non-final text items

`NonInitialItemsFile` - a name of the file containing a list of non-initial text items

`NonInitialPrefixFile` - a name of the file containing a list of non-initial prefixes

`InputCharSet` - string representation of the Character Set that will be used while reading the input files

`OutputFile` - a name of the output file for writing the compiled sentence boundary recognizer resources

Returns:

`true` if this operation was succesfull, or `null` otherwise

ComputeBoundaries

public int[] **ComputeBoundaries**(char[] InputText,
sbToken[] Tokens)

Returns an array of indices of sentence boundaries.

Specified by:

ComputeBoundaries in class AbstractSentenceBoundaryRecognizer

Parameters:

`InputText` - array containing the input text.

`Tokens` - an array of tokens (@see de.dfki.lt.runtime.sentenceBoundary.sbToken.) Please note that any tokenizer can be used via converting the character-position information in the original tokens into the corresponding de.dfki.lt.runtime.sentenceBoundary.sbToken objects which solely record the start/end character positions of the tokens

Returns:

an array of integers indicating indices of character positions which represent sentence/utterance boundaries. If the input text or token list is of length 0 then `null` will be returned.

ConvertSproutTokens

public static sbToken[] **ConvertSproutTokens**(TokenIterator pTI)

This method converts tokens generated by SProUTs' tokenizer into de.dfki.lt.runtime.sentenceBoundary.sbToken objects which are required as input to the method for computing sentence/utterance boundaries

Parameters:

`pTI` - a token iterator (@see de.dfki.lt.sprout.runtime.tokenizer.TokenIterator)

Returns:

an array of sbToken objects or `null` if the pTI is `null`.

---

isWhitespace

public boolean **isWhitespace**(char c)

Checks if an input character is a whitespace.

Parameters:

`c` - input character

Returns:

`true` if the character is a whitespace, or `false` otherwise

---

configureLogger

public boolean **configureLogger**(java.lang.String LoggerPropertyFile)

Configure the logger according to the Logger property file

Parameters:

`LoggerPropertyFile` - the property file used for configuring the logger

Returns:

`true` if the configuration was successfull, or `false` otherwise.

---

setBasicLogger

public void **setBasicLogger**()
Activates the default basic logger (standard output)

---

setSimpleLogger

public void **setSimpleLogger**(java.lang.String FileName)
Configures the logger: Messages are redirected to the input file, and the level is set to 'INFO'.
Parameters:
`FileName` - - a name of the file for messages etc.

JAVA DOCUMENTATION FOR THE CLASS sbToken

---

de.dfki.lt.sprout.runtime.sentenceBoundary

Class sbToken

java.lang.Object
└ **de.dfki.lt.sprout.runtime.sentenceBoundary.sbToken**

---

public class **sbToken**
extends java.lang.Object

This class implements Token type used by the Sentence boundary recognition. It just contains the start/end character position of the tokens. In this way, any tokenizer may be used. Note that this type is also used for representing text spans (not only tokens).

Method Summary	
int	**getEnd**() Returns end position of the token
int	**getStart**() Returns start position of the token

void	**set**(int start, int end) Sets the current token start/end position to given values

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

getStart

public int **getStart**()

Returns start position of the token

Returns:

start position of the token

---

getEnd

public int **getEnd**()

Returns end position of the token

Returns:

end position of the token

---

set

public void **set**(int start,
int end)

Sets the current token start/end position to given values

Parameters:

`start` - initial character position

`end` - final character position
