Learning the Meaning of Words

Architectural Design

January 14th, 2011

by Till Maurer, Stefan Helfrich, Simon Loew, Karsten Knuth, and Eva König

supervised by Birgit Schwarz

project provided by
Afra Alishahi

Class diagram

The class-diagram designed according to the MVC architectural pattern. Therefore the *model*, the *view* and the *controller* are separated and organized individually.

The model contains all data needed in the application domain.

The view will be implemented as a GUI. Since this implementation is quite dependent on a programming language, the design of the view is purposely left unspecific. For testing purposes the GUI could be replaced by an automated interface which can test the program without involving user input.

The controller supervises the flow of data through the application and is responsible for computations.

Class diagram

Program
-controller : Controller
-model : Model
-view : View

Model
-situations : List<Situatio...
-parameters : Parameters
-wordFilter : WordFilter
-focusedWords : List<Wo...
-wordTable : List<Word>

WordFilter
-learningThreshold : Double
-frequencyThreshold : Inte...

Situation
-utterance : List<Wor...
-scene : List<Meanin...

Parameters
-utteranceFile : File
-imageDB : Folder
-logFile : File
-additiveNoise : Double
-substractiveNoise : Dou...

Meaning
-stringRep : String
-image : Image
+Meaning(stringRep : String)
+Meaning(stringRep : String, image : Im...

Word
-stringRep : String
-frequency : Integer
-states : Map<step : Integer, Map<meaning : Meaning, probability : Double>>
+Word(stringRep : String)
+getLastState()
+setState(iteration : Integer, state : Map<Meaning, Double>)

Observable
-observers : List<Observer>
+setChanged()
+notifyObservers()

<<Interface>>
Observer
+update(Observable)

org.jdom.Element

org.jdom.Document

LogReader
-situationFactory : SituationFactory
-logFile : File
+buildWord(element : Element) : Word
+buildDistribution(elements : List<Element>) : Map<Meaning, ...
+buildParameters()
+buildSituations()
+buildWords()

LogWriter
-logFile : File
-document : org.jdom.Document
-wordElementMap : Map<Word, Element>
-createWordElement(word : Word) : Element
-createMeaningElement(map : Map<Meaning, Double>) : El...
+flush()
+update(Observable)

Controller
-view : View
-model : Model
-currentIteration : Integer
+setParameters(utteranceFile : File, substractiveNoise : Double, logFile : File, additiveNoise : Double, learningThreshold : Double, imageDB : Folder)
+preprocess()
+process()
+loadDefault()
+saveDefault()
+changeDataPanel(panelId : Integer)
+addWordToFocusedPanel(word : String)
+removeWordFromFocusedPanel(word : String)
+dragWordFromUtteranceToFocused(word : String)
+goForward(stepSize : Integer)
+goBackward(stepSize : Integer)
+goToFirstIteration()
+goToLastIteration()
+exportImage(image : Image)
+savePreprocessedInput()
+saveCompleteResults()
+loadPreprocessedInput()
+loadCompleteResult()
-getWordTable()
-filterWords()

Preprocessor
+preprocess(model : Model)

Parser
+parse(file : File) : List<List<String>>
-readFile(file : File) : List<String>
-split(sentences : List<String>) : List<List<Strin...
-stem(words : List<List<String>>) : List<List<S...

Situation Factory
+createSituations(utterances : List<List<String>>, model : ...
-createWord(stringRep : String) : Word
-createMeaning(stringRep : String) : Meaning
-addNegativeNoise(negativeNoise : Double)
-addAdditiveNoise(additiveNoise : Double)

Processor
+runAlgorithm(situations : List<Situati...

LearningAlorithm
+run(situations : List<Situation>)
-step(Situation)

<<Interface>>
Algorithm
+run(situations : List<Situati...
-step(Situation)

org.jfree.data.xy.DefaultXYDataset

View
-controller : Contro...

GraphicalUserInterface
-Panels : List<Panel>
-activePanel : Panel
+changePanel()
+updateActivePanels()

Panel
+updatePanel()

Menubar

SidePanel

DataPanel
-sidePanel : SidePanel
-navigation : NavigationP...

NavigationPanel

ParameterPanel

ScenePanel

WordComprehensionScorePanel

WordsLearnedOverTimePanel

WordTablePanel

org.jfree.chart.ChartFactory
+createXYLineChart(xy : DefaultXYDataset) : JFreeChart

org.jfree.chart.JFreeChart

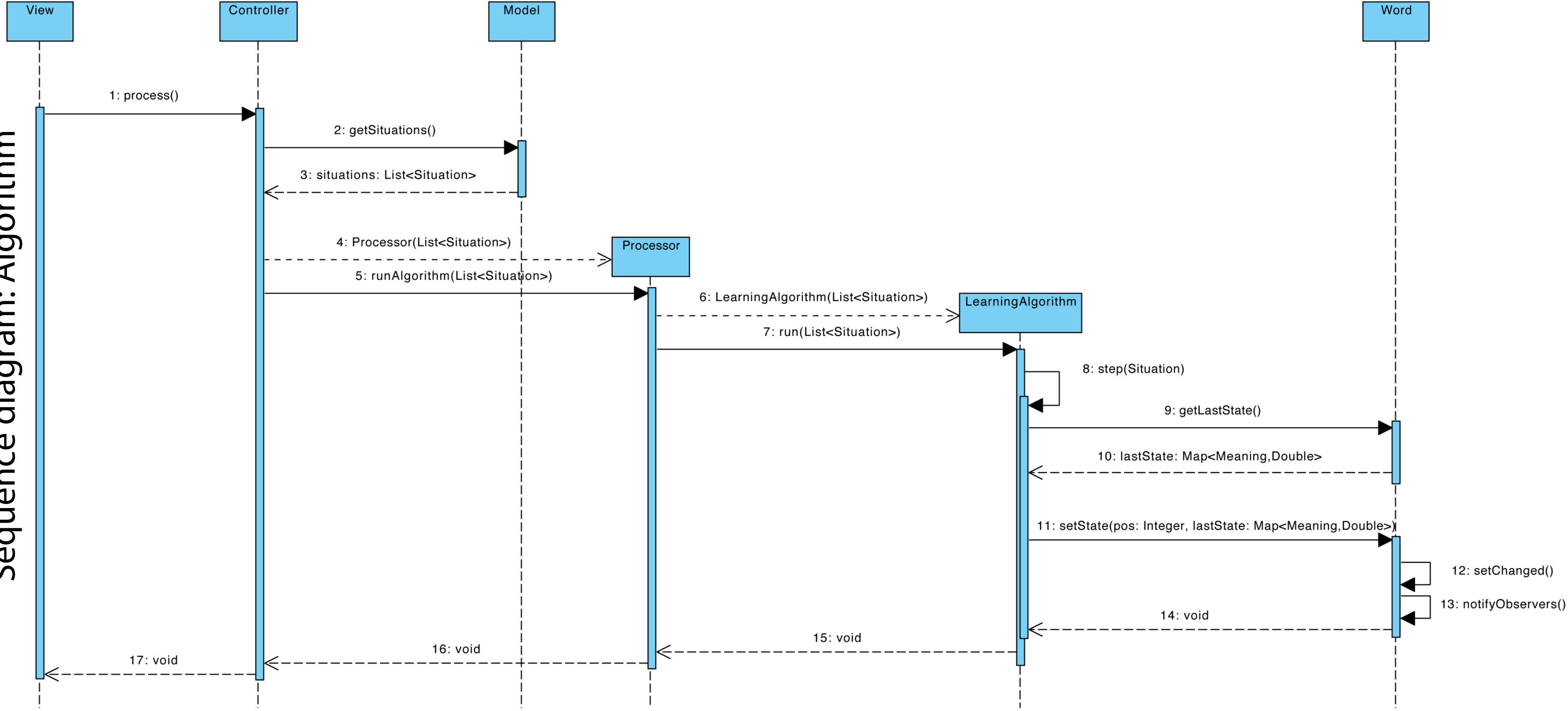
<<instantiate>>
<<instantiate...
<<use>>
instantiates all panels

Sequence diagram: Algorithm

This sequence diagram shows the processing of the situations with a learning algorithm. In a previous preprocessing step, a list of situations has been compiled from the utterance input file in respect to the set parameters. A situation represents an event observed and thus has a list of of words (from one utterance) and its corresponding scene (a list of meanings).

The processing of the situation is triggered by the user by clicking on the 'build' button in the GUI. This action is passed to the Controller, which requests the preprocessed situations from the Model. Then the Controller instantiates the Processor, which then instantiates the LearningAlgorithm. For each situation, the Learning Algorithm calls the step method, which performs one iteration of the algorithm. Here, for each word of the situation its state (which represents the possible meanings of the word with their associated probabilities) is updated.

Sequence diagram: Algorithm



Sequence diagram: Preprocessing

This sequence-diagram describes the preprocessing of the input data to build up the data-objects required by the algorithm to simulate the learning process.

When the program is started, it creates the three modules needed i.e. *model* (the data-model), *controller* (main controller for communication between view and model as well as for computing), and *view* (presentation-layer, usually the GUI) and links them.

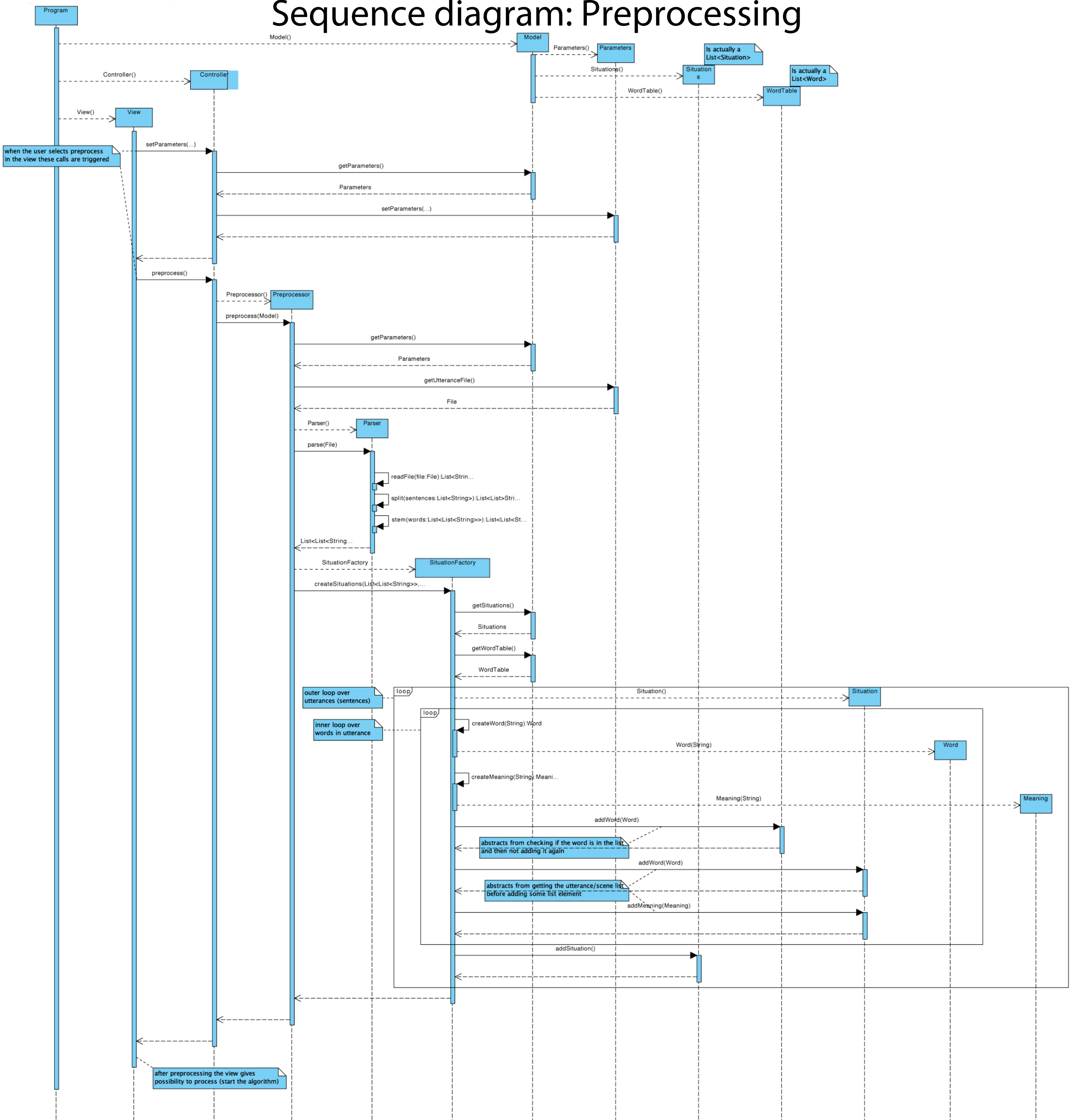
The diagram begins after selection of all parameters by the user in the view, which represents the parameter-input-screen (see use-cases/GUI-design). Then the user selects the preprocess button and the view triggers the method setParameters(...) and preprocess() in the controller.

In the setParameter(...)-method the controller stores the given parameters in the model.
In the preprocess()-method the controller generates a parser which takes the given utterance-file and parses it to a list (sentences of the utterance-file) of a list (words of the utterances) of stemmed words.

This list of list of strings is returned to the controller who generates a situation-factory to create the list of situations. The controller further fills the word-table for performance reasons. The situation-factory loops with an outer loop over the sentences from the utterance-file and creates for each sentence a new situation-object. Within this loop the inner loop iterates over the words in each sentence and creates for each word an appropriate word-object and its associated meaning-object. These objects are stored in a situation-object (the word-objects are stored additionally in the word-table as mentioned above).

When the inner loop finishes, the situation-object is stored in a situations-list.
Finally, the preprocessing is done and the user can start the algorithm (see algorithm-sequence-diagram).

Sequence diagram: Preprocessing

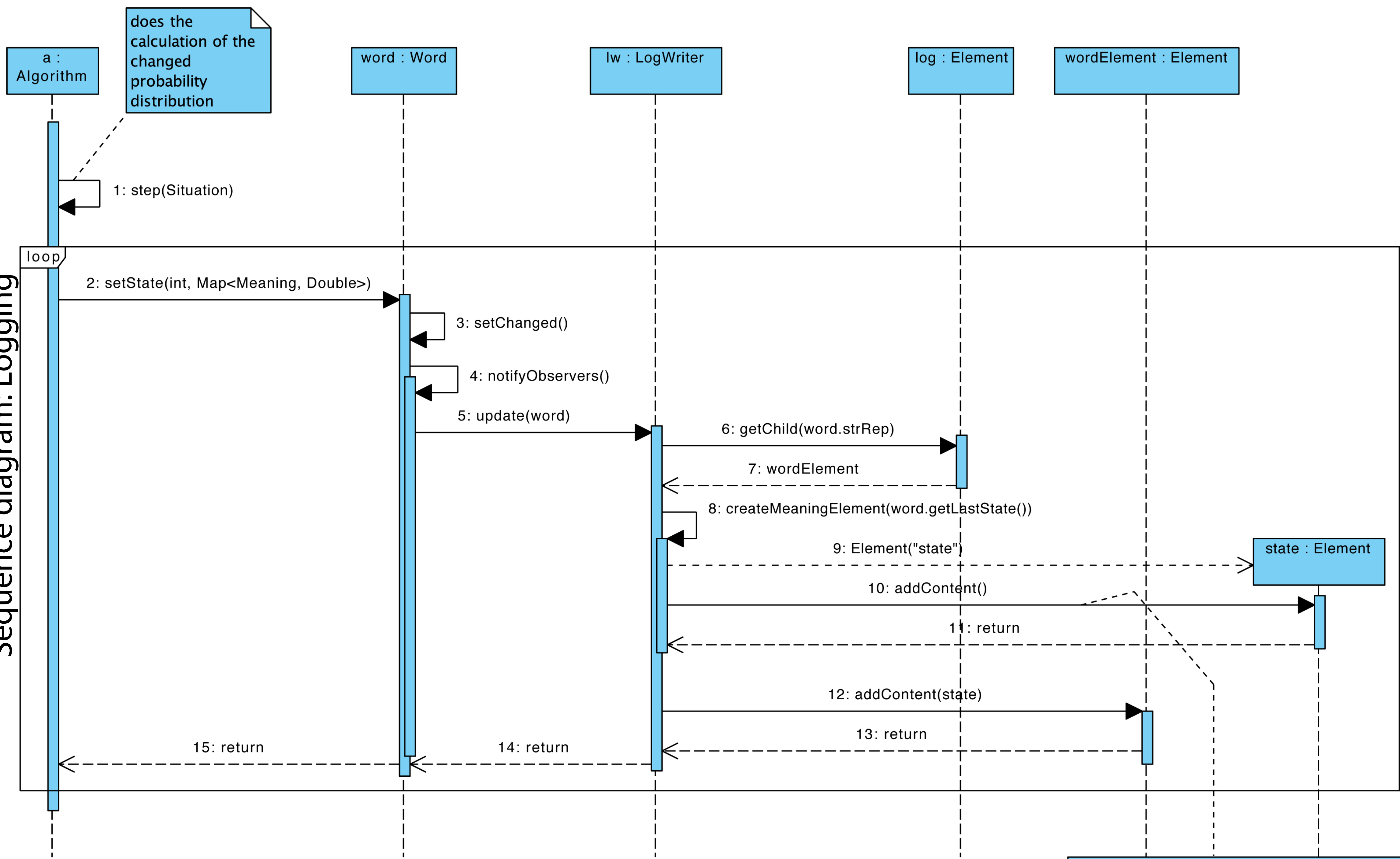


Sequence diagram: Logging

This sequence diagram depicts the process of writing the information storing classes (model classes) to XML. However, this process is not only logging but also a kind of serialization. This is due to the fact that that later on model classes are instantiated and initialized with the data contained in the XML file. For the purpose of reading and writing XML we plan to use the external API JDOM.

Before **p** (a preprocessor) calls *run()* on **a** (an implementation of an algorithm) the parameters and situations have been generated by the according classes. In the temporal order the parameters and situations are logged before the algorithm starts execution. Since the process of logging those classes to XML is similar to the one depicted in this sequence diagram, this part is omitted.
Furthermore, the word that is to be updated has occurred before and therefore no Element has to be explicitly created but only updated with the new probability distribution.

Sequence diagram: Logging



Activity diagram

This activity diagram shows an overview over the whole program. The user starts the program and can choose between three options. First he/she can start a new program run by setting an utterance input file and other parameters. This input is then preprocessed to situations by the system. Second, the user can directly load situations, which he/she has preprocessed in a previous program run. Now, in both cases, the situations are processed by the execution of the learning algorithm. As a third option, the user can load a complete program run, which he/she has previously compiled and processed. Now, in all three cases, the user can analyse arbitrary iterations of the algorithm by navigating through them. For each iteration, the user can view certain features, e.g. the image scene corresponding to the situation, the parameters set, word comprehension scores, a word table, and the proportion of words learned over time. The latter features can be exported and saved as images or tables. The user can further conduct two panels of word where he/she can conveniently view their features. Finally, the user can exit the program.

Activity diagram

Program User

System

new run

set parameters

preprocessing

load preprocessed situations

load preprocessed situations

save preprocessed situations

build

process situations

load complete run from file

execute algorithm

write results to log file

view 'Word Table'

view 'Scene'

view 'Word Comprehension Score'

view 'Parameters'

view 'focused words' and 'utterance words'

view 'Words Learned over Time'

navigate through iterations

export data

add word

delete word

move words

exit program