# Requirements Tiamat

Ayrton Vercruysse

November 13, 2012

## Contents

# 1 ASTs

## 1.1 Making ASTs

**Description:** Every piece of code used in the IDE will be directly linked to an AST. For this matter there has to be a possibility to create new ASTs from scratch. To create new AST trees we will need to implement the different parts of which an AST can exist. These different parts of an AST will be called a Node. We will start with a list of possible nodes with which basic programs can be made. Later extra kinds of nodes can be added to the AST. A superclass Node will be implemented which will organise the tree structure by providing operations like getParent(), wich gets the parent of a Node, setChild(oldChild, newChild) which will set a child of a node getChilderen() to get the childeren of a Node. By using these operations new trees can be created.
**Priority:** High
**Status:**

## 1.2 Deleting ASTs

**Description:** When we have an excisting AST and parts of this AST became irreleveant there should be a possibility to remove this AST. The removing of an AST, in total, or only a part of an AST will happen by replacing this AST by the Placeholder node. The actual replacing can happen should happen in the Node class, by using a function setChild(oldChild, newChild).
**Priority:** High
**Status:**

## 1.3 Manipulating ASTs

### 1.3.1 Replacing parts of ASTs

**Description:** This can be done on a similar way as the deletion of ASTs, or parts of AST this will happen by replacing the current AST by a new AST which only contains the Placeholder node.
**Priority:** High
**Status:**

## 1.4 Rendering ASTs to text

**Description:** To be able to pass the constructed code to the AmbientTalk interpreter we have to convert the AST to an understandable format for the interpreter. The interpreter works with a textfile containing AmbientTalk code. This means that the ASTs have to be converted to text. Converting an AST to

a text should be a function within each Node, calling the same function recursively on all of it's childeren.
**Priority:** High
**Status:**

# 2 Templates

## 2.1 Templates creating ASTs

**Description:** For convience we will make use of templates. Templates will be frequently used parts of code that will be saved. The way templates will be stored in the memory will be by an AST. Every time a template is used withing the program a copy of this AST will be made and inserted into our current program.
**Priority:** High
**Status:**

## 2.2 Creating templates from XML

**Description:** When having an XML file containing a structure to create new Templates a function will be implemented that creates, from the XML file, a new AST, which can be used as a Template within the program.
**Priority:** High
**Status:**

## 2.3 Creating functions on the spot

**Description:** Whenever a new function (or variable) is created within the program a link to call this function will be added to the menu's. This will be the call of this function.
**Priority:** High
**Status:**

## 2.4 Create templates from functions

**Description:** When a newly created function is often used by the user and he wants to save this function, the possibility to write this function, in correct XML, to the existing XML file with all templates should be forseen.
**Priority:** High
**Status:**

# 3 Interface

## 3.1 Views

### 3.1.1 Colorcoding

**Description:** As in many languages the use of colorcoding should be done here aswell. In our implementation we will make use of two sorts of colorcoding. First we will use colorcoding for the block around the code, wich will suggest the type of the code (e.g. Template, functionname, variablename), and on the other hand the usual colorcoding of special keywords will be implemented by changing the color of the names itselve.
**Priority:** High
**Status:**

### 3.1.2 Codefolding

**Description:** The use of AST should gives us the possibility to easily fold in certain parts (read piece of the AST) of our code. This can be done by just not rendering certain nodes, and it's childeren, of the AST.
**Priority:** High
**Status:**

### 3.1.3 Move code

**Description:** By using the possibility of deleting parts of ASTs and saving parts of ASTs a copy-pasty system, or even moving of code to places that make sense belongs to the possibilities.
**Priority:** High
**Status:**

## 3.2 Gestures

## 3.3 Selection of code

**Description:**
**Priority:** High
**Status:** **Description:** When starting on a piece of code by using the pinch-to-zoom function the selected part of code should be extended. This can be done by enlarging the selected area from the current AST to the parent of this AST.
**Priority:** High
**Status:**

### 3.4 90 degrees turning for comments

**Description:** When a piece of code is selected, and afterwards this piece of code gets turned around for 90 degrees, this piece of code should be commented out. Commented code should not be written to the textfile when this is called on this AST.
**Priority:** High
**Status:**

### 3.5 Fast scrolldown

**Description: When we want to scroll down a longer piece of code we can use the scroll with two fingers getures to get us scroll faster.**
**Priority:** High
**Status:**

## 4 Evaluating code

### 4.1 Writing code to textfile

**Description:** To make use of the AmbientTalk app we need a textfile with the AmbientTalk code. We can make use of the toText() function of each node to translate our current AST to plain text, and afterwards we will need to write this text, into a textfile.
**Priority:** High
**Status:**

### 4.2 Call external AmbientTalk app

**Description:** The external AmbientTalk app, which makes use of the textfile from the previous - insert number- should be called. To get this app called we will implement a button on the screen, which first gets the textfile created and afterwards calls the AmbientTalk app.
**Priority:** High
**Status:**

### 4.3 Return to TIAMAt after evaluating code

**Description:** The AmbientTalk app will evaluate the program. After the evaluation of this program and returning the result we should get back the our application to create the possibility to edit our current program
**Priority:** High

**Status:**

# 5 Features

## 5.1 Extra interface for comments

**Description:** An extra interface can be created in which we can store extra comments. This interface should only be called on object which could get any use of extra comments.
**Priority:** High
**Status:**

## 5.2 Speaking comments

**Description:**
**Priority:** High
**Status:**

## 5.3 Selector for Java classes

**Description:**
**Priority:** High
**Status:**