Requirements Tiamat

Ayrton Vercruysse

December 2, 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(), which gets the parent of a Node, setChild(oldChild, newChild) which will set a child of a node and getChilderen() to get the childeren of a Node. By using these operations new trees can be created. The types of nodes that will be implemented are: Placeholder, Begin, Block, Definition, Value, Argument List, Function, Function Call, Function Definition, Operation, Table, Table Call Table Definition.
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. This possibility will be, as said in 1.1 implemented in the Node class, by means of the setChild(oldChild, newChild) function.
Priority: High
Status:

1.3 Replacing parts of ASTs

Description: When new Nodes will be created it's childeren, if there are any, will be make by Placeholders. When making a function for example, we'll implement the operation, but the two operands will be initialized with a Placeholder node. To be able to change these Placeholders to more relevant Nodes we should be able to replace existing Nodes by other Nodes, in this example maybe a Value, or an other Operation. This can be done on a similar way as the deletion of ASTs, by making use of the replaceChild(oldChild, newChild) method.
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 convenience 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. These templates can be frequently used functions, but mainly these will be the ASTs representing basic elements of the language like the if-then-else sturcture, when-discovered,...
Priority: High
Status:

2.2 Creating templates from XML

Description: To keep a clear view on what template functions we have, and make it easy for users to add templates, without having to know the implementation of the entire program we will keep the Templates in a separate XML file. When having an XML file containing the 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 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 will be provided. This means that whenever the program is terminated functions can be stored inside the XML file.
Priority: High
Status:

3 Interface

3.1 Rendering of nodes

Description: Each type of node has to have a screen representation. For each node we need a function that displays the Node on the screen, and that recursively calls the display function of all children of this node. We won't implement these displays within the separate types of nodes, discribed in 1.1, but for each node we will implement a separate display function. This with as goal that we can re-use our AST implementation without having to dead with the program specific display possibilities.
Priority: High
Status:

3.2 Lay-out engine

Description: We will make a render manager, this manager will render all nodes and make sure they are organised on a ordered way. The render mananger will use indetion to organise nodes across the screen. **Priority:** High
Status:

3.3 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 means that we can easily re-use a made variable or have an easy way to call an earlier created function.
Priority: High

Status:

3.4 Buttons

3.4.1 Menu's

Description: To keep track of all templates or calls that can be made we will need a menu that gives us the possibility to insert templates in to the code. We will use one a menu with some submenu's, devided for each kind of templates. These can be provided functions self implemented function, used variables,...
Priority: High
Status:

3.4.2 Run button

Description: Thanks to requirements 4.2 we can evaluate made code with an external AmbientTalk app. To do this we'll add a button to the main screen which will first make sure requirement 1.4 is done, the creation of a textfile with the current code, and later on run the AmbientTalk app, as described in requirement 4.2. **Priority:** High
Status:

3.4.3 Delete button

Description: To be able to delete some code, as described in 1.2 we need a way to select what code will be deleted. Herefore we will implement a delete button on the screen, which deletes the current selected piece of code. This can be done by pressing this button or by dragging the selected code to this place.
Priority: High
Status:

3.5 Views

3.5.1 Colorcoding keywords

Description: As in many languages the use of colorcoding should be done here aswell. We will try to achieve the same colorcoding as being used in the official AmbientTalk IDE in Eclipse.
Priority: Low
Status:

3.5.2 Colorcoding types

Description: Next to the colorcoding of keywords we will implement colorcoding on the types of words. This means that we will give Placeholders different colors, just like VariableNames,...
Priority: Low
Status:

3.5.3 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: Low
Status:

3.5.4 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: Low
Status:

3.6 Gestures

3.6.1 Selection of code

Description: To be able to replace, delete, move,... code we need a possibility to select certain parts of the code. This will be done by a tap or doubletap gesture. The selected code will be marked and it will be possible to makes changes to this code.
Priority: High
Status:

3.6.2 Pinch-to-zoom

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:

3.6.3 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: Low
Status:

3.6.4 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: Low
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: Low
Status:

5.2 Speaking comments

Description: A nice feature is to add spoken comments to our program. The possibility to add spoken comments will be integrated in the extra interface for spoken comments, announced in requirement 5.1. There will be added extra buttons to record comments and play these comments.
Priority: Low
Status:

5.3 Selector for Java classes

Description: Within the AmbientTalk language it's possible to make use of a Java class selector. Adding this feature to this implementation will make sure that our project becomes a better alternative to the pc one.
Priority: Low
Status:

5.4 Multiple tabs

Description: If we want to make bigger programs it would be easy to be able to have multiple tabs. Certainly when big programs exsist of multiple files. To make this possible we would need to save multiple AST trees and implement a possibility to switch between the different tabs.
Priority: Low
Status:

5.5 Saving files

Description: As we will implement a write to XML to save new Template into an XML file we could use this procedure to save an entire program in XML,

and later rebuild our AST and on this way reconstruct our program.
Priority: Low
Status: