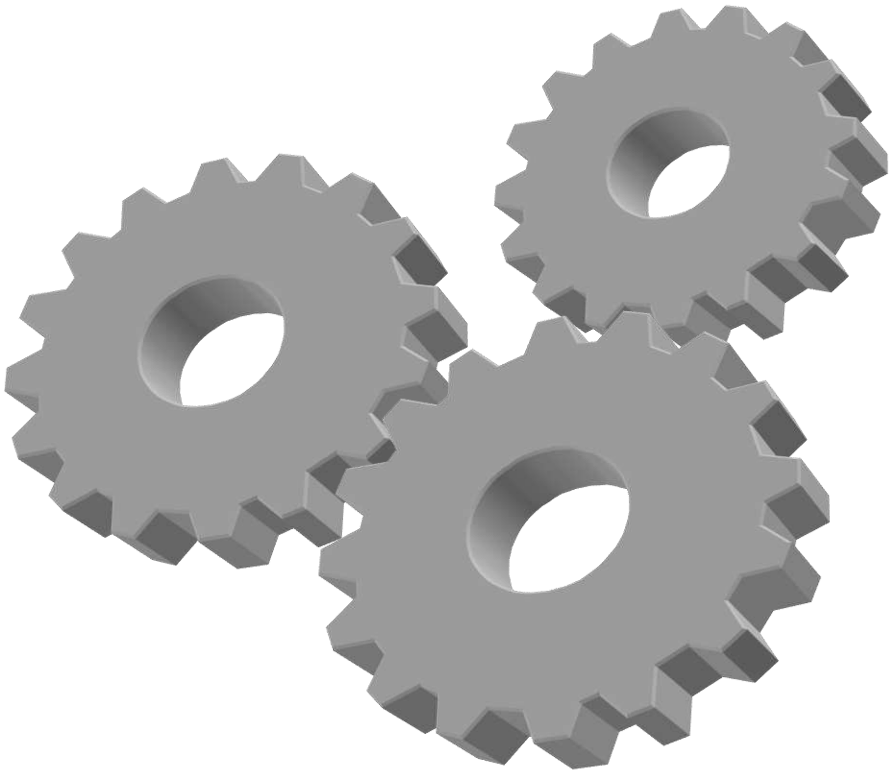
Vanga

[1.0.0]
Tervel Atanassov



[VANGA User Manual]

This manual will explain the major ideas behind VANGA. It will illustrate these ideas by referring to the sample VANGA configuration to drive the system calculator GUI app.

Revision History

Version	Date	Revision Description
1.0.0	8/04/2010	Initial Version

Table of Contents

Contents

1. Introduction

VANGA recognizes screens of applications by evaluating the image that a user of the application would normally see on the screen. The usual approach for driving GUI applications is to access their components based upon window handles and other such elements which the window manager sees. Because of the wide disparity of approaches to drawing themselves taken by applications, VANGA foregoes this approach in favor of only evaluating the final product, that which is seen by the user on the screen. Structurally, the definition of the screens is to look for unique anchor points which are treated as tokens in the definition of a visual language. The tokens compose lexemes which in turn compose other lexemes. VANGA examines the screen and tries to match it one of its defined lexemes. Once a screen matches a lexeme, it along with the spatial distribution of the tokens is passed to a user defined handler whose purpose is to manipulate the screen. The actual manipulation of the screen such as clicking on it is done through a wrapper of the standard java Robot interface.

Purpose

This document attempts to introduce the reader to the major concepts of configuring VANGA. Namely, it guides the reader through the definition of the grammar required to identify the keys of the system calculator application. It continues by guiding him through the actions he can perform within his custom script object such as navigating the mouse and clicking buttons.

Related Documents

This document should be included in a file called getting-started-configuration.zip. You can download this package from the documents section of the VANGA project page: http://sourceforge.net/projects/vanga/

Conventions

Use relevant content here.

Problem Reporting Instructions

Should you encounter any bugs with VANGA, please help us make the product better by recording them on the project home: http://sourceforge.net/projects/vanga/

2. Overview

The VANGA configuration file consists of three major sections corresponding to the major phases in the processing of an application's screen. Screens which are to be processed originate from providers. Think of these as the abstract providers of screenshots. VANGA expects that implementations defined in this section will return a single screenshot. In turn, this screenshot needs to be classified in order to be processed. The automaton which performs the recognition is defined in the grammar section. Screens which are successfully classified are identified by the root lexeme which matched them. This information is used to find the appropriate handler which can then interact with the application which produced the original screenshot.

Providers

Individual providers are defined in the **<providers>** section of the VANGA configuration file. Two key pieces of information need to be provided for each provider. The obvious one is the **class** attribute. This tells VANGA which Java™ class it needs to create. The second one is the unique name of the provider. This is specified via the **name** attribute. Names identify specific providers and therefore need to be unique for the scope of the file. Additionally, one provider needs to be designated as the default one. This is done via the name separator symbol “|” (bash pipe symbol) followed by the special name “Default”. Unless directed otherwise, VANGA gets its screenshot from the default provider. This means that the very first provider which is queried is the default provider.

Apropos from the sample configuration…

The sample configuration contains a single provider. Here is what its definition looks like:

```
<provider name="Apps|Default" class="com.vanga.screen_acquisition.ScreenDump">
    <arg name="accept">Calculator</arg>
    <arg name="unrecognized_location">unrecognized</arg>
</provider>
```

It has been given two names, Apps and Default. The first is arbitrary and is provided in order to remind the user that he is free to name his providers as he wishes. The second tells VANGA that – unless instructed otherwise by a Handler – this is the provider to retrieve screenshots. The Default provider is the first one to be queried when VANGA starts up.

Grammar

The grammar is defined in the **<grammar>** section of the VANGA configuration file. This section is composed of two sub-sections: **<tokens>** and **<lexemes>**.

The tokens section defines the foundation of the grammar because individual tokens represent actual images which serve as anchors for the lexeme super structure which is erected above them. As a result, tokens are atomic, i.e. they cannot be compositions of other tokens or lexemes. The token node is configured defined through a number of attributes. The following table lists the possible attributes:

Attribute Name	*Possible Values*	*Description*
error-threshold	Numeric	An integer representing the percentage of pixels in the image which can differ, AND the two images to

		still be considered to match. For example if the search image is 10x10 pixels and the error-threshold has been set to 15. That means that this image will be considered to match in the search image any time a region is found which has more than 85 pixels which match per the criteria specified by the tolerance attribute.
colorspace	bw	The images will be matched in the binary, black/white color space.
	gray	The images will be matched in the grayscale color space.
	rgb	The images will be matched in the RGB color space.
file	A path in the file system referring to an image file.	The file which contains the template image which will be searched in subsequently provided images. *Note: In the case that multiple files need to be loaded, the framework aids the user in the following way:* *The file attribute has been specified as: "path/name.png"* *All files with the name "path/name-N.png" will be loaded as separate tokens.* *This helps the user because he can add other incarnations of the same template image without having to update the grammar for each one.*
match_type	exact	The controlled image will be matched exactly to the test image. *Note: Using this will be the fastest way to find tokens; however, even a single pixel variation will cause the template image to not be found.*
	inexact	A correlation search (http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient) will be performed in order to find the controlled image in any images subsequently passed to the token. *Note: This is the slowest way of finding sub-*

		images; however it is tolerant to differences in the two images.
	tolerant	This is will perform an exact search of the controlled image in the image provided during operation, but a certain amount of differences will be accepted for a match. *Note: Refer to the attributes tolerance, and error-threshold.*
name	Alphanumeric	This is the unique name by which this token will subsequently be referred to in the configuration.
tolerance	Numeric	The allowed numeric difference between two pixels in the image. If the two pixels differ by less than this amount they are considered to be a match.
type	image	This token controls an image which it will try to find in images which are given to it during operation.
	ocr	This token will perform OCR on the regions of images given to it during operation.

Apropos from the sample configuration…

```
<token name="button-upper-left-token" type="image" match_type="tolerant"
colorspace="gray" tolerance="5" error-threshold="1" file="images/button-upper-left.png"/>
```

The xml node above defines a token which will load the image file with a relative path “images/button-upper-left.png”. It will search for this image in subsequently provided images and consider that it has found a match any time less than 5% of the pixels differ by more than 1 in the gray-scale color space. This token will be known in the rest of the configuration by its name “button-upper-left-token”.

```
<token name="content" type="ocr" />
```

The xml node above defines a token which will perform OCR on whatever regions it is given. This token will be known in the rest of the configuration by its name “content”.

The lexemes section defines what valid screens or their sub-regions look like. A lexeme is a definition which specifies how individual lexemes and tokens come together to form a particular composite structure. For example, suppose we had defined what the upper left and lower right corners of some button look like, and these definitions took the form of tokens. Using these tokens, we could define a button as an upper left corner token AND lower right corner token to the bottom right of it. This definition would be a simple lexeme. In turn, we could use it to define more complex lexemes such as row of buttons, and an array of buttons such as those seen on a calculator.

Like a token definition, a lexeme definition contains a number of properties which need to be specified in order to construct coherent grammars in VANGA.

name	alphanumeric	This is the unique name by which this token will subsequently be referred to in the configuration.

The actual composition is accomplished via the definition of child nodes of the parent lexeme. As indicated earlier these can be either other lexemes or tokens. The following table defines the valid parameters of a child lexeme definition:

name	alphanumeric	This is the unique name of the sub-lexeme which we expect to be contained.
occurrences	numeric	A number specifying the number of occurrences of the sub-lexeme
	v	The specified sub-lexemes will be distributed vertically.
	h	The specified sub-lexemes will be distributed horizontally.
	+	The specified sub-lexemes will be one or more.
	*	The specified sub-lexemes will be zero or more.
	!	The specified sub-lexemes need to overlap in the direction they have been defined (h or v) to be laid out.
optional	boolean	If true, the sub-lexeme does not necessarily need to exist. Default is false, i.e. all sub-lexemes which have not been specified as optional need

		to be matched in order for the containing lexeme to be considered as matched.
positions	(m,n),(j,k)…	If the containing lexeme is considered to be an MxN grid, this specifies at which positions of the grid the sub-lexeme can exist.
synthetic	lexemeA lexemeB operator	This specifies that the sub-lexeme is synthetic, i.e. it is a composition of other sub-lexemes which have already been identified within the current lexeme. The control language gives the way these lexemes are composed. For example: A B + means that the sub-lexeme is the union of the already found lexemes A and B.

Apropos from the sample configuration…

```
<lexeme name="screen">
    <token positions="(0,0)" name="screen-upper-left-token"/>
    <token positions="(1,1)" name="screen-lower-right-token"/>
    <lexeme synthetic="screen-upper-left-token screen-lower-right-token +" name="active-area"/>
</lexeme>
```

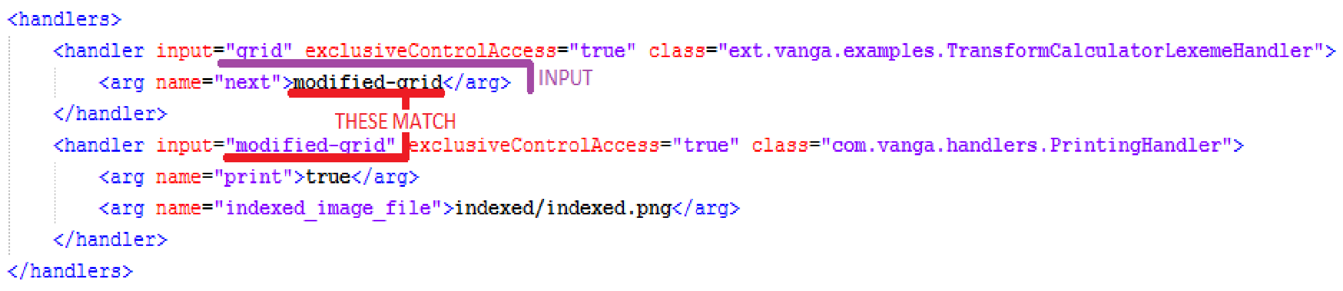
The above node defines a lexeme called "screen" which will be identified by the token "screen-upper-left-token" in the upper left corner (implied by the attribute "position=(0,0)") and the token screen-lower-right-token in the lower right corner (implied by the attribute "position=(1,1)").

Once these two sub-tokens have been identified, the union of their containing rectangles will be given to the lexeme called "active-area". This lexeme refers to an OCR token. The effect of this is that once the screen has been identified, OCR will be performed on its containing rectangle. This will enable VANGA to read the result of the calculator's calculations.

Handlers

Handlers are defined in the **<handlers>** of the VANGA configuration file. There are several configuration parameters which are shared among all handlers. The first and most obvious is the implementing class. This is specified in the class attribute of each node. Next is the name of the identified lexeme which this handler will process. This is specified in the “input” attribute of the handler node. Finally, if the user is planning to perform incremental processing which will be distributed over the methods of several handlers, he can chain these handlers by specifying the name of the transformed lexeme. This is accomplished via the argument with the name “next”. Essentially, by operating on the screen, each handler can modify it thereby transforming it into another lexeme. The next argument names this newly produced lexeme so that the framework can pass it along for further processing to another handler which has the name as part of its input attribute.

Apropos from the sample configuration…



```
<handlers>
    <handler input="grid" exclusiveControlAccess="true" class="ext.vanga.examples.TransformCalculatorLexemeHandler">
        <arg name="next">modified-grid</arg>
    </handler>
    <handler input="modified-grid" exclusiveControlAccess="true" class="com.vanga.handlers.PrintingHandler">
        <arg name="print">true</arg>
        <arg name="indexed_image_file">indexed/indexed.png</arg>
    </handler>
</handlers>
```

The configuration above defines two handlers. The first accepts an input lexeme called “grid” and produces a new lexeme called “modified-grid” which is – in turn – accepted by the second handler.

3. Instructions for Running VANGA

Prerequisites

1. Maven. Being the latest word in modernity, VANGA utilizes the latest – at the time of this writing – craze, Apache Maven. You should have a passing

familiarity with what Maven does. Although VANGA has been tested and runs with Maven 2.2+, you should try to run it on the latest version.

2. LINUX distributions Only. Because the VANGA Java Native Interface under Linux uses shared memory to get the screenshots, should you want to compile/develop you need to get the dev packages. Under Ubuntu you need to run the following:

```
sudo apt-get install libx11-dev x11proto-xext-dev libxext-dev
```

3. TBD

Running the example configuration

1. *Optional*. If you want to build the vanga-core component on your local machine, you need to get the vanga-core from the SourceForge svn repository via this command:
   ```
   svn co https://vanga.svn.sourceforge.net/svnroot/vanga/vanga-core
   ```
   Then you would build the code with the command:
   ```
   mvn install
   ```
2. Retrieve example sources from the SourceForge svn repository via this command:
   ```
   svn co https://vanga.svn.sourceforge.net/svnroot/vanga/vanga-examples
   ```
3. Navigate a system shell to the newly downloaded vanga-examples directory and build the required example files via this command:
   ```
   mvn install
   ```
4. Run your system's calculator app, e.g. under gnome linux it is called gcalctool.
5. In the vanga-examples directory, run the example configuration via this command
   ```
   mvn exec:java
   ```

4. Reference

Error messages and causes

VANGA utilizes the java.util.logging package. By default it logs to the console. If you want to create a custom logging configuration, or pipe output to the currently existing logging infrastructure of your application, you should search the web for the appropriate configuration.

There are a number of error messages. They were all written to be self-explanatory and explaining them in this document would imply that we didn't do a good job with the messages themselves. However, here are some common ones whose resolution is not straightforward, but whose existence could greatly complicate your interaction with VANGA and impinge upon – what was intended to be – an otherwise hassle-free experience.

- *INFO: There is a collision within the lexeme "grid" for sub-lexeme "button-row". It will be moved from position (0,0) to (1,0).* This is message from the sample configuration. As the names of the lexemes suggest, it refers to the following configuration item:

```
<lexeme name="grid">
    <lexeme occurrences="1" name="screen"/>
    <!-- +v means one or more in the vertical direction.-->
    <!-- +h means one or more in the horizontal direction.-->
    <!-- replacing + with * has the same implications but for zero or more.-->
    <lexeme occurrences="+v" name="button-row"/>
</lexeme>
```

  In this configuration it is of no consequence. The reason why it is brought up in this section is that – as the log statement indicates – the VANGA framework is making an assumption that you want your visual lexeme "grid" to be laid in the vertical direction. Should you not want such a layout, you need to refer to the previous sections which describe how to control the layout of lexemes.
-

Cross references to other operations

Use relevant content here. Use relevant content here. Use relevant content here. Use relevant content here. Use relevant content here. Use relevant content here. Use relevant content here. Use relevant content here. Use relevant content here. Use relevant content here.

Appendix A - Glossary

Title	Title

Appendix B – Index

Title	Title
